2002-02-20

ISO Technical Committee ISO/TC 184, *Industrial automation systems and integration*,
Subcommittee SC4, *Industrial data*

Guidelines for the development of abstract test suites, edition 2

This standing document replaces
the Guidelines for the development of abstract test suites (SC4 N536).

ISO TC 184/SC4 Secretariat
DISA/JECPO Center for Standards
10701 Parkridge Blvd
Reston
VA 22091
USA
email: sc4sec@cme.nist.gov

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO/TC184/SC4 Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the ISO TC184/SC4 Secretariat at the address given below.

SC4 standing documents specify procedures, methods, and guidelines for the development of SC4 standards for industrial data. SC4 standing documents, together with additional documents such as templates and checklists, can be obtained from the Internet:

<http://www.nist.gov/sc4/www/necsdocs.htm>

ISO TC 184/SC4 Secretariat
DISA/JECPO Center for Standards
10701 Parkridge Blvd
Reston
VA 22091
USA
email: sc4sec@cme.nist.gov

Contents

Page

Figures

Tables

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

This standing document was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data*.

The following standing documents provide guidelines for developing International Standards produced by ISO/TC 184/SC4:

— ISO/TC 184/SC4 Organization handbook;

— SC4 Quality manual;

— SC4 Supplementary directives.

The following standing documents provide additional guidelines for developing parts of ISO 10303 “Product data representation and exchange”:

— Guidelines for application interpreted construct development;

— Guidelines for application interpreted model development;

— Guidelines for the development and approval of STEP application protocols;

— Guidelines for the development of mapping specifications;

— Guidelines for the development of mapping tables.

This standing document cancels and replaces the Guidelines for the development of abstract test suites (SC4 N536).

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation of product data and for exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout their life cycle. This mechanism is suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases, and as a basis for archiving.

This International Standard is organized as a series of parts, each published separately. The parts of ISO 10303 fall into one of the following series: description methods, integrated resources, application interpreted constructs, application protocols, abstract test suites, implementation methods, and conformance testing. The series are described in ISO 10303-1.

The purpose of this standing document is to provide methods and procedures for the development of abstract test suites for conformance testing implementations of ISO 10303 application protocols. The intended audience for this document is application protocol developers and abstract test suite developers. Abstract test suite development is an extension of the application protocol development process. As such, abstract test suite development is typically performed by the respective ISO 10303 application protocol development project. Specifics on style, format, required text, and other presentation details of the abstract test suite are provided in the SC4 Supplementary directives – Rules for the structure and drafting of SC4 standards for industrial data. SC4 requirements regarding development of abstract test suites have changed several times since the publication of the first edition of this document. The reader is urged to refer to the SC4 Organization handbook and the Cumulative list of resolutions for current mandates.

This is the second edition of this standing document. There are four primary changes from the first edition:

— Test purposes that are derived directly from the application protocol's application reference model and application interpreted model according to the steps spelled out in annex A of this document are no longer repeated in the abstract test suite. Repeating requirements from the application protocol is an opportunity to introduce errors, and is a maintenance issue while the application protocol is under development. The expense for abstract test suite developers does not outweigh the benefit gained by executable test suite developers. Executable test suite developers can explicitly derive and document the lists of test purposes at the time the executable test suite is created.

— The requirement for maintaining cross references between the lists of test purposes and the test cases has been removed. These cross references were difficult for abstract test suite developers to create and maintain, without significant added benefit.

— Text describing SC4 mandates on projects to develop abstract test suites and to govern the level of coverage thereof has been removed in this edition.

— Text describing style, format, required text and other presentation details has been deleted. That text may be found in SC4 Supplementary directives – Rules for the structure and drafting of SC4 standards for industrial data.

An exception to the rule that examples must appear in 10 point type has been made in various places throughout this document so that editors can see exactly how the information being presented should appear. Examples that show specific formats or layouts to be used in SC4 standards are distinguished from other text by a light grey background. Examples have been reformatted from the source AP to match the new clause structure for mapping specifications.

Guidelines for the development of abstract test suites, edition 2

1 Scope

This standing document specifies content of ISO 10303 abstract test suites. It also specifies methods and procedures that SC4 projects follow to develop abstract test suites.

The following are within the scope of this document:

— development process for creating an abstract test suite suitable for use in testing implementations of ISO 10303 application protocols based on ISO 10303-21 and ISO 10303-22;

NOTE Procedures and algorithms for developing test purposes, while not of central importance to this document, are also included.

— procedures and algorithms for developing verdict criteria;

— content and structure of abstract test cases;

— suggested approaches for structuring and creating abstract test cases;

— suggested approaches for validating an abstract test suite and its component abstract test cases.

The following are outside the scope of this document:

— methods or procedures for developing conformance requirements of application protocols;

— methods or procedures for partitioning application protocols into conformance classes;

— methods or procedures for evaluating interoperability between implementations based on different application protocols;

— requirements on SC4 projects to produce abstract test suites;

— requirements on the level of coverage of abstract test suites produced by SC4 projects;

— formatting, layout, and required text for an abstract test suite.

2 Normative References

The following documents contain provisions which, through reference in this text, constitute provisions of this standing document. For dated references, subsequent amendments to, or revisions of, such publications do not apply. However, editors of SC4 standards are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. For undated references, the latest edition of the document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO 10303-1:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles*.

ISO/TR 10303-12:1997, *Industrial automation systems and integration — Product data representation and exchange — Part 12: Descriptive methods: The EXPRESS-I language reference manual.*

ISO 10303-21:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure.*

ISO 10303-22:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 22: Implementation methods: Standard data access interface.*

ISO 10303-31:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 31: Conformance testing methodology and framework: General concepts.*

ISO 10303-32:1998, *Industrial automation systems and integration —- Product data representation and exchange — Part 32: Conformance testing methodology and framework: Requirements on testing laboratories and clients.*

ISO 10303-34:2001, *Industrial automation systems and integration — Product data representation and exchange — Part 34: Conformance testing methodology and framework: Abstract test methods.*

ISO/IEC8824-1:1994, *Information technology —- Open systems interconnection —- Abstract syntax notation one (ASN.1) —- Part 1: Specification of basic notation.*

The following SC4 standing documents contain provisions, which, through reference in this text, constitute provisions of this standing document. At the time of adoption, the revisions of the documents indicated were valid. All documents are subject to revision, and users of this standing document are encouraged to investigate the possibility of applying the most recent revisions of the documents indicated below.

ISO TC 184/SC4 N534:1997, *Guidelines for application interpreted construct development.*

ISO TC 184/SC4 N532:1997, *Guidelines for application interpreted model development.*

ISO TC 184/SC4 N535:1998, *Guidelines for the development and approval of STEP application protocols.*

ISO TC 184/SC4 N1190:2001, *Guidelines for the development of mapping specifications, edition 2.*

ISO TC 184/SC4 N533:1997, *Guidelines for the development of mapping tables.*

ISO TC 184/SC4 N1191:2001, *SC4 Supplementary directives - Rules for the structure and drafting of SC4 standards for industrial data.*

NOTE A list of current SC4 standing documents and other material to be used by developers of SC4 standards is available from the Internet:

http://www.nist.gov/sc4/www/necsdocs.htm

3 Terms, definitions, and abbreviations

3.1 Terms defined in ISO 10303-1

For the purpose of this standing document, the following terms defined in ISO 10303-1 (repeated below for convenience) apply.

3.1.1
application interpreted model (AIM)
information model that uses the integrated resources necessary to satisfy the information requirements and constraints of an application reference model, within an application protocol

3.1.2
application reference model (ARM)
information model that describes the information requirements and constraints of a specific application context

3.1.3
conformance class
subset of an application protocol for which conformance may be claimed

3.1.4
exchange structure
a computer-interpretable format used for storing, accessing, transferring, and archiving data

3.2 Terms defined in ISO 10303-31

For the purpose of this standing document, the following term defined in ISO 10303-31 (repeated below for convenience) applies.

3.2.1
abstract test case (ATC)
a specification, encapsulating at least one test purpose, that provides the formal basis from which executable test cases are derived. It is independent of both the implementation and the values

3.2.2
abstract test method
the description of how an implementation is to be tested, given at the appropriate level of abstraction to make the description independent of any particular implementation of testing tools or procedures, but with sufficient detail to enable these tools and procedures to be produced

3.2.3
basic tests
limited tests performed to determine whether it is appropriate to perform thorough testing

3.2.4
conformance testing
the testing of a candidate product for the existence of specific characteristics required by a standard in order to determine the extent to which that product is a conforming implementation

3.2.5
executable test case
an instantiation of an abstract test case with values

3.2.6
fail (verdict)
a test verdict given when the observed test outcome demonstrates non-conformance with respect to either the test purpose or at least one of the conformance requirements in the relevant standard(s)

3.2.7
implementation under test (IUT)
that part of a product which is to be studied under testing, which should be an implementation of one or more characteristics of the standard(s) based on a given implementation method

3.2.8
inconclusive (verdict)
a test verdict given when the observed test outcome is such that neither a pass or a fail verdict can be given

3.2.9
pass (verdict)
a test verdict given when the observed test outcome gives evidence of conformance to the conformance requirement on which the test purpose is focused and is valid with respect to the relevant standard(s) and with respect to the PICS

3.2.10
postprocessor
a software unit that translates product information from an independent public domain product data format to the internal format of a particular computer system

3.2.11
preprocessor
a software unit that translates product information from the internal format of a particular computer system to an independent public domain product data format

3.2.12
Protocol Implementation eXtra Information for Testing (PIXIT)
a statement made by the client which contains or references all of the information (in addition to that given in the PICS) related to the IUT and its corresponding SUT, which will enable the testing laboratory to run an appropriate test suite against that IUT

3.2.13
test purpose
a precise description of an objective which an abstract test case is designed to achieve

3.2.14
test realisers
an organisation which takes responsibility for providing, in a form independent of the clients of a testing laboratory and their IUTs, a means of testing IUTs

3.2.15
(test) verdict
a statement of "pass", "fail", or "inconclusive" concerning conformance of an IUT with respect to an executable test case and the abstract test case from which it was derived

3.2.16
verdict criteria
information defined within an abstract test case which enables the testing laboratory to assign a verdict

3.3 Other terms and definitions

3.3.1
abstract test suite (ATS)
part of ISO 10303 that contains the set of abstract test cases necessary for conformance testing of an implementation of an application protocol

NOTE Adapted from ISO 10303-1.

3.3.2
application element
application object, attribute of an application object, or assertion of a relationship between two application objects

3.3.3
application protocol (AP)
part of ISO 10303 that specifies an application interpreted model satisfying the scope and information requirements for a specific application

NOTE 1 This definition differs from the definition used in open system interconnection (OSI) standards. However, because ISO 10303 will not be used directly with OSI communications, no confusion should arise.

NOTE 2 Adapted from ISO 10303-1.

3.3.4
coverage
the percentage of all possible test purposes that are satisfied by input data specifications from at least one abstract test case

3.3.5
expected output specification (EOS)
a presentation of the output from a conforming implementation to be expected as the result of a conformance test

3.3.6
implementation method
part of ISO 10303 that specifies a technique used by computer systems to exchange product data that are described using the EXPRESS language

NOTE Adapted from ISO 10303-1.

3.3.7
minimal entity set
the set of AIM entity instances which shall be present in any instantiated model of a given AP

3.3.8
minimal object set
the set of instances of application objects which shall be present in any instantiated model of a given AP

3.4 Abbreviations

For the purpose of this standing document, the following abbreviations apply.

AE application element (see 3.3.2)

AIM application interpreted model (see 3.1.1)

AP application protocol (see 3.3.3)

ARM application reference model (see 3.1.2)

ATC abstract test case (see 3.2.1)

ATS abstract test suite (see 3.3.1)

EOS expected output specification (see 3.3.5)

IUT implementation under test

PIXIT protocol implementation extra information for testing (see 3.2.12)

4 Abstract test suite development

This document defines the content of and procedures for developing an abstract test suite (ATS) for use in conformance testing implementations of ISO 10303 application protocols (APs). These guidelines are based on the requirements set forth in the ISO 10303-30 series parts on conformance testing:

— ISO 10303-31 provides an overview of conformance testing concepts. High-level considerations specified therein guide the entire conformance testing process and the development of test suites.

— ISO 10303-32 covers the relationships, requirements, and roles of test laboratories and their clients. ISO 10303-32 discusses the PIXIT that depends in part on aspects of the abstract test suites.

— ISO 10303-34 provides abstract test methods for conformance testing implementations of application protocols using the exchange structure defined by ISO 10303-21 or using the standard data access interface ISO 10303-22.

NOTE 1 Requirements of the abstract test methods in ISO 10303-34 affect the content of abstract test suites.

Any characteristic or combination of characteristics called out in an International Standard can be the basis for conformance testing.

The standardization of abstract test suites corresponding to ISO 10303 APs facilitates establishment of consistent conformance testing programmes world-wide. An ISO 10303 abstract test suite is composed of a set of abstract test cases. Each abstract test case specifies input data to be provided to the implementation under test, along with the associated verdict criteria. Abstract test cases define the necessary tests in a form that is applicable to all processors to be tested.

NOTE 2 In order to perform conformance testing, testing laboratories convert abstract test cases into executable test cases that are in a form that can be executed on the system under test.

NOTE 3 AP projects are encouraged to develop executable test cases during the development of the AP. Informative executable test cases may be included in an informative annex of the ATS.

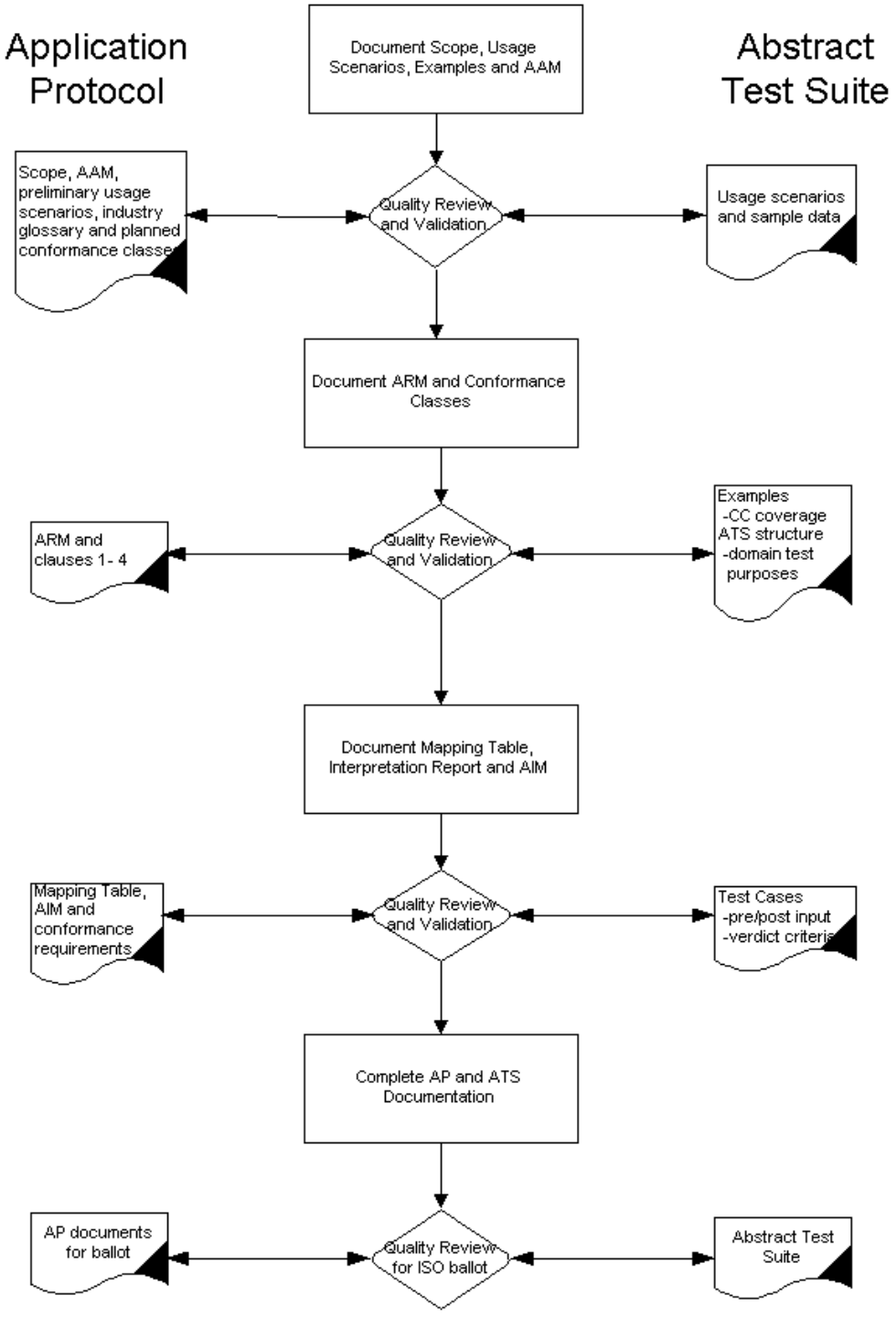

Figure 1 – Combined AP and ATS development process

Ideally, ATS development proceeds in parallel with the development of the related AP. The AP and ATS development process can be viewed as one process where the deliverables are documented in two different standards (see Figure 1). Upon completion of the various stages of AP development, the

quality review and validation of those AP components leads to the development of related ATS components. The usage scenarios that are created during the development and review of the scope and information requirements of an AP provide the structure for the test cases of the ATS. The derivation and review of application element (AE) test purposes is a good application reference model (ARM) validation exercise. The usage tests employed in the validation of the ARM can be used for the preprocessor input specifications for abstract test cases.

Some example abstract test cases should be developed before the AP is released as a committee draft. The development of these abstract test cases aids in validating the AP and in preparing the AP project for developing the ATS.

The overall process for creating the abstract test suite is as follows:

a) Generate and review the set of test purposes from the AP that describes the testing objectives identified as important and relevant by AP domain experts. Important test objectives are those derived from usage scenarios and sample test data from real industry applications. Make any needed changes to the AP and iterate until the AP/ATS development team accepts the test purposes.

b) Plan the abstract test cases based on usage scenarios created for the AP and understand their relationship to the test purposes.

c) Generate test case input data specifications and verdict criteria for preprocessor and postprocessor implementations in parallel, driven by the same input application semantics.

d) Construct a single abstract test case for preprocessors and postprocessors using the parallel sets of input data specifications and verdict criteria.

e) Add the abstract test case to the abstract test suite.

f) Assess how well the abstract test suite covers the test purposes as a whole and by each conformance class.

g) Loop back to step b until the desired (or mandated) level of test purpose coverage is achieved.

The primary components of a test suite are the specific test purposes, general test purposes and verdict criteria, test cases, and supporting annexes. The development of these components is discussed in the following clauses.

5 Specific test purposes

Test purposes are statements of testing objectives related to some aspect of an ISO 10303 application protocol. Test purposes are developed by breaking down the conformance requirements of an ISO 10303 AP into a finite set of explicit statements of test objectives to be covered by the ATS. Conformance requirements arise from specific clauses of an AP as well as from other parts of ISO 10303. Test purposes typically describe application information requirements, data structures, constraints or operations on instantiated models, and model data.

An abstract test suite specifies inputs to an implementation under test that are designed to cover the set of test purposes derived from the AP. The challenge of test purpose development is to produce a set of test purposes that is finite and manageable in size, but is also comprehensive and explicit enough to avoid misinterpretation of the Standard during the development of test cases and subsequent conformance testing.

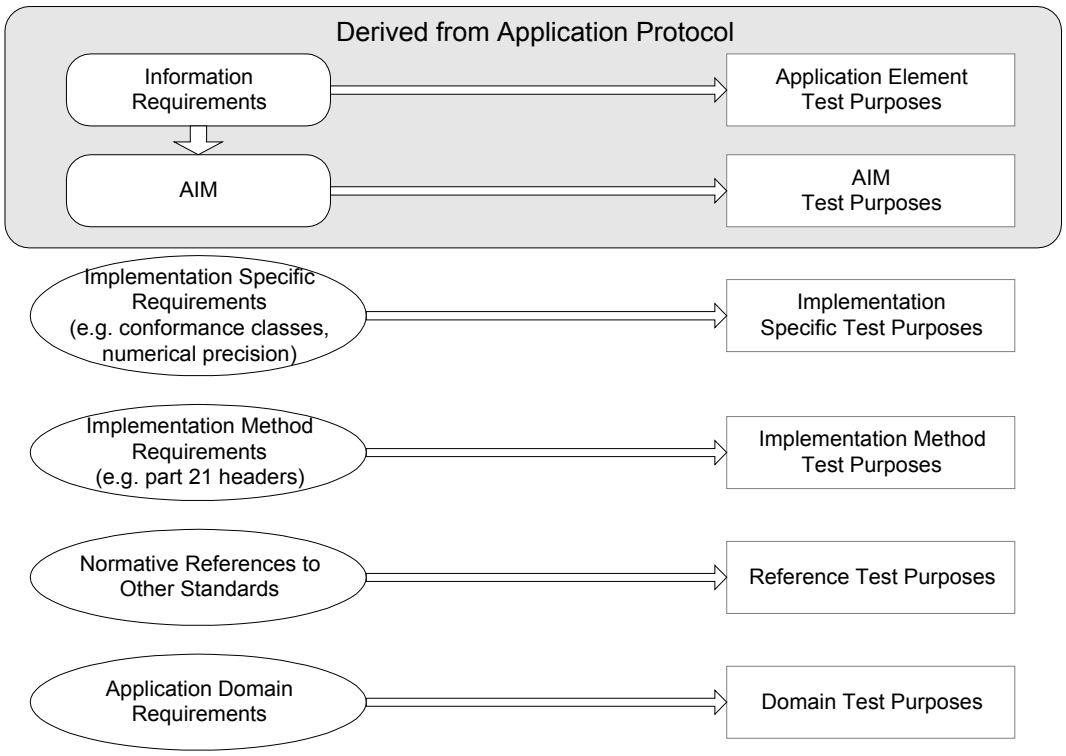

Figure 2 – Test purpose sources

Figure 2 illustrates several primary sources of test purposes. There may be other sources of test purposes not covered here. Test purposes are either derived algorithmically from the AP specification or are explicitly documented by the ATS developers. The lists of test purposes that are derived from the information requirements and interpreted model in clauses 4 and 5 of the AP shall not be documented explicitly in the ATS. Only those specific test purposes not derived algorithmically from the AP are documented in clause 4 of the abstract test suite. Clause 4 contains test purposes from the following sources: implementation methods, domain requirements, referenced standards, and other sources as needed. Specific test purposes developed and documented in clause 4 of the ATS often must be associated with specific verdict criteria that are developed to verify the satisfaction of the specific test purpose. Specific verdict criteria are documented in the appropriate test case. See 7.3.4 and 7.4.3 for more on development and documentation of specific verdict criteria.

5.1 Application element test purposes

Application elements are the application objects, attributes, and application assertions that represent the normative information requirements of the AP. An AE test purpose is implicitly associated with each application element. AE test purposes are atomic-level statements of the information requirements of the AP.

AE test purposes are a useful tool for validating the AP ARM. Although AE test purposes are not explicitly documented in the ATS, they should be derived once a complete ARM has been developed for the AP. The AE test purposes should be reviewed by the AP team to ensure that they accurately reflect the intended scope and semantics of the ARM. Any needed modifications can be made to the ARM and the resulting test purposes reviewed to ensure that the requirements are correct.

The meaning of a test purpose derived from the information requirements is given as follows: the IUT (either preprocessor or postprocessor) shall preserve the semantic associated with the application element from which the test purpose was derived. This implies that the semantics of the application

element are preserved by the IUT between the input and output of a test according to the reference path specified in the mapping table of the AP.

AE test purposes are useful in developing preprocessor input and generating specific verdict criteria for the outputs of both preprocessors and postprocessors.

The methods for deriving and formally specifying AE test purposes are provided in annex A.

5.2 Application interpreted model test purposes

AIM test purposes can be derived from the long form of the AIM (annex A of an AP). They represent the structure of the EXPRESS information model. AIM test purposes are atomic-level statements of the implementable information model of the AP.

As with AE test purposes, AIM test purposes can be used to validate the AIM of the AP and to quantify the exact coverage in the ATS of the APs information model. AIM test purposes are also not explicitly restated in the ATS. AIM test purposes should be derived and reviewed by the AP team as a validation exercise once the AIM is complete.

The meaning of a test purpose derived from the AIM EXPRESS schema is given as follows: the postprocessor shall accept the input in accordance with the AIM EXPRESS structure and AP mapping corresponding to this test purpose. This implies no violations of any constraints (e.g. where rules or global rules) that apply to the AIM element.

Two high-level conformance requirements that pertain to all APs characterize the bounds of the set of AIM test purposes:

— Only those constructs specified in the AIM shall be produced by a preprocessor or accepted by a postprocessor implementation of the AP.

NOTE Preprocessor implementation may produce user-defined entities, but they are only checked for conformance to ISO 10303-21 syntax.

— All entities, types, and their associated constraints identified in a particular conformance class shall be supported by postprocessor implementations of the AP. Treatment of options and default values shall conform to the AIM.

AIM test purposes are particularly useful in developing test case data for postprocessor implementations, which accept inputs in an ISO 10303 form (i.e., ISO 10303-21 or ISO 10303-22) corresponding to the structure of the EXPRESS information model.

Since testing all possible instantiated models is impractical, reasonable heuristics should be adopted to guide the selection of representative models in the actual test case data. These heuristics focus conformance testing on a useful set of representative test data that is valid for the application domain and worth the cost of testing. The set of AIM test purposes should be reviewed and test purposes that are not a high priority for testing should be removed.

EXAMPLE 1 STRING-valued attributes can exist in the AIM structure that are not explicitly mapped or constrained in the AP. These attributes are not a high priority for testing.

EXAMPLE 2 If an AIM entity attribute is defined as being "for information only", then the test purpose that addresses that attribute is a likely candidate for removal. An example is the attribute b_spline_curve.self_intersect. This attribute is a flag that indicates whether the given b-spline curve intersects itself. The attribute is listed as being for information only so that if there is a discrepancy between the curve data and this flag, the curve data takes precedence.

EXAMPLE 3 When units of measurement are brought in from ISO 10303-41, the entire set of units comes as a single group. In any particular AP, such as ISO 10303-203, many of those measures such as luminous_intensity_measure, are likely to be unused. Test purposes associated with such unused units shall be removed.

5.3 Other test purposes

Other test purposes can be developed as a result of conformance requirements arising from other aspects of the AP or ISO 10303. These other test purposes can be derived from requirements that are implicit within the scope of an AP, from other parts of ISO 10303, or from referenced standards. All other test purposes are explicitly documented in the ATS, and shall have at least one associated general or specific verdict criterion. Other test purposes are uniquely identified in the ATS. Other test purposes are positively phrased statements of requirements expressed in complete sentences.

5.3.1 Domain test purposes

Domain test purposes are other test purposes that arise from requirements that are implicit in the AE of an AP but are important enough to make into an explicit test purpose. Domain test purposes are optional and shall not add requirements beyond those in the associated AP.

Domain test purpose requirements can exist at a finer level of detail than is explicitly provided in the AEs of the AP. Domain test purposes can also describe explicit combinations of elements in particular configurations that are characteristic within the AP application domain.

EXAMPLE ISO 10303-202 has an application object Organization that is responsible for zero, one or many Drawings, zero, one or many Drawing_sheets, and zero, one or many Product_versions. There is an additional requirement documented in the AP (e.g. as a global rule in the AIM) that an Organization must have responsibility for at least one of these to be included in an exchange. This last requirement is not expressed in the structure of the information requirements in the AP and does not yield a derived AE test purpose. ISO 10303-302 includes the following domain test purpose:

other13 Each Organization has responsibility for at least one Drawing, one Drawing_sheet, or one Product_version

5.3.2 Implementation methods test purposes

Implementation method test purposes are other test purposes specific to a particular implementation method. An AP can have requirements specific to an implementation method as expressed in its normative annex C, titled "Implementation method specific requirements."

EXAMPLE ISO 10303-21 exchange structure header requirements can be a source of test purposes for abstract test suites designed to test implementations that use the exchange structure.

5.3.3 Referenced standards test purposes

Standards represented as normative references in the AP can contribute additional other test purposes. Based on knowledge of the domain, abstract test suite developers shall determine whether the content of a normatively referenced standard places additional conformance requirements on implementations of the AP. Referenced standards may include standards outside of ISO 10303.

EXAMPLE The normative references clause of ISO 10303-201 cites ISO 3098-1: 1974 - Technical Drawings - Lettering - Part 1: Currently used characters. In ISO 10303-201 5.2.2.20, the draughting_pre_defined_text_font states that "the information on pre-defined text fonts which shall be supported by all implementations of this part of ISO 10303 is given in ISO 3098-1." This requirement suggests that one or more test purposes address the referenced pre-defined font.

6 General test purposes and verdict criteria

General test purposes are statements of requirements that apply to every abstract test case, to all preprocessor test cases, or to all postprocessor test cases. General test purposes are stated in natural language as requirements on an implementation of the ISO 10303 AP. General test purposes are developed from requirements that can be implicit in the test purposes of clause 4. All general test purposes are uniquely identified in the ATS.

EXAMPLE 1 The following are examples of general test purposes:

g1 The output of an IUT shall preserve all the semantics defined by the input model according to the reference paths specified in the mapping table defined in clause 5 of ISO 10303-203.

g2 The output of a preprocessor shall conform to the implementation method to which the IUT claims conformance.

General verdict criteria are the means for evaluating whether the general test purposes are met. General verdict criteria are stated as natural language statements that can be evaluated as PASS, FAIL, or INCONCLUSIVE and formalized as necessary to ensure they are unambiguous. General verdict criteria shall be evaluated as a part of every executable test case to which they apply. Each general verdict criterion includes a reference to its associated test purpose in parenthesis at the end of the test purpose.

A general verdict criterion is associated with a general test purpose, and verifies that its associated test purpose(s) is (are) satisfied across multiple abstract test cases. All general verdict criterion are uniquely identified in the ATS.

EXAMPLE 2 The following are examples of general verdict criteria:

gvc1 The semantics of the input model are preserved in the output of the IUT according to the reference paths specified in the mapping table defined in clause 5 of ISO 10303-203 (g1).

gvc2 The output of a preprocessor conforms to the implementation method to which the IUT claims conformance (g2).

General test purposes and associated general verdict criteria are documented in clause 5 of the abstract test suite to avoid unnecessary repetition and encourage clarity.

7 Abstract test cases

The abstract test suite shall contain abstract test cases that apply to both preprocessors and postprocessors. Preprocessor and postprocessor input specifications with the same input semantics are documented together in the same abstract test case. Some abstract test cases may have more than one postprocessor input specification with the same input semantics as the preprocessor input specification.

Abstract test cases may be sequenced in any manner the developers deem useful. It may be useful to order abstract test cases by AP conformance class. A table shall be included in annex A of the abstract test suite that identifies for each conformance class the set of abstract test cases that apply.

EXAMPLE 1 If an abstract test suite has no conformance classes but has an identified set of basic tests, then the first abstract test cases would be the basic tests, followed by the other individual abstract test cases.

NOTE No subjective qualification of abstract test cases shall be documented within the abstract test suite. Terms such as "real-world" or "synthetic" test cases shall not be used even though such distinctions may be made during the development of the abstract test cases.

The value of the abstract test suite lies in the abstract test cases, each of which tests some functionality of the AP. The abstract test cases provide implementers and test laboratories with the information they need to test products for conformance.

It is critical that each test case provides a realistic test of one or more test purposes. Therefore, abstract test cases should be based on realistic use of the information within the AP's scope. Test cases should exploit real-world data sets taken from industrial applications. Another technique is to refer to the AAM for usage scenarios from which the ARM and AIM were developed. Any usage scenarios used for developing abstract test cases that is not documented as part of the AP shall be documented in an informative annex of the abstract test suite.

Developing abstract test cases is not a simple, well-defined process. Given that exhaustive testing is not possible, the application protocol developers and implementers must decide what to test and how thoroughly to test it. In general, the greater the impact a proposed abstract test case provides in detecting and preventing product model data exchange errors, the more value it will provide as part of an abstract test suite. APs provide a formal structure with fully defined semantics for exchange. In spite of this, problems with exchange are certain to occur. If, in the standard or product development process, a test case is generated that is known to be useful in detecting or preventing a problem in product model data exchange, that test case should be included in the abstract test suite.

The following steps provide an approach to building test cases:

a) Identify those application elements that can be combined into a single abstract test case. Use whatever approach makes sense in combining the requirements. Develop specific test purposes, if needed. Ensure that all application elements in a test case belong to the same conformance class.

EXAMPLE 2 It is often sensible to identify the minimal object set and the minimal entity set, by definition required to exist in every abstract test case.

b) Develop the abstract test cases.

   1) Develop a preprocessor input specification.

   2) Develop corresponding postprocessor input specification(s).

   3) Develop specific verdict criteria, if needed.

c) Evaluate test suite coverage and iterate.

d) Group the abstract test cases according to conformance classes.

It may also be useful to work in the other direction for some test cases. That is, if a useful test case input specification has been identified through some means, then the test purposes covered by the specification(s) should be identified, specific verdict criteria developed as needed, and a complete abstract test case developed.

Each abstract test case addresses both preprocessor and postprocessor implementations. While the same semantics apply to both, the input specifications for the two types are different. The abstract test case shall contain the input specifications and suitable verdict criteria for both preprocessor and postprocessor tests. Because a given application element can map to more than one AIM construct, an abstract test case may contain more than one postprocessor input specification for completeness.

EXAMPLE 3 Assume an AP has an application element b_spline_curve. Using the integrated resources (ISO 10303-42 in this case), that would map to eight types of b-spline curves. In an abstract test case designed to evaluate whether a processor could handle b-spline curves, the preprocessor input specification would include a b-spline curve. However, the equivalent postprocessor input specification would come in four forms, one for each of the types of b-spline curve allowed.

The input specification is a concise description of the input data requirements for the abstract test case. Each abstract test case will contain the specification for the input data to be used when developing and running an executable version of that abstract test case. The input specification defines the concepts and relationships to be tested without mandating specific data values, except where the test purpose itself mandates specific values. If suggested values are provided as part of an input specification, they shall not represent any actual person, company, or organization.

An abstract test case is made up of the following components; optional and repeatable components are noted. These components are described in more detail in the following subclauses. Execution sequence and extra details may pertain to the test case as a whole or to the preprocessor or postprocessor only. These additional components are described last.

a) Abstract test case identification

   1) Test case summary;

   2) Execution sequence (optional);

   3) Extra details (optional).

b) Preprocessor

   1) Test purposes covered;

   2) Input specification;

   3) Verdict criteria;

   4) Constraints on values (optional);

   5) Execution sequence (optional);

   6) Extra details (optional).

c) Postprocessor (may be more than one)

   1) Test purposes covered;

   2) Input specification;

   3) Verdict criteria;

   4) Execution sequence (optional);

   5) Extra details (optional).

7.1 Abstract test case identification

Each abstract test case shall be documented as a separate subclause of the abstract test case clause. The subclause number serves as the unique abstract test case identifier. The subclause title serves as

the title of the abstract test case. The title may be any useful and appropriate informal reference to the test case.

7.2 Test case summary

In order to assist users in understanding the scope of each abstract test case, a test case summary shall be included in each abstract test case. The purpose of the test case summary is to provide an explanation of the high-level goal of the abstract test suite. The test case summary shall be the first paragraph of the abstract test case. Test case summaries shall be concise and are typically no more than a few sentences.

EXAMPLE The following is an example of a test case summary "This test case checks the ability of an IUT to correctly instantiate an advance b-rep solid model. The minimal amount of configuration management information is included to produce a conforming model, but none of that data is checked in this test case."

7.3 Preprocessor testing

Each abstract test case shall contain a subclause that contains the input specification and other details necessary for preprocessor testing. The preprocessor subclause shall be made up of three or more subclauses: covered test purposes, input specification, verdict criteria, and optionally, constraints on values, execution sequence, and extra details.

7.3.1 Preprocessor test purposes covered

The list of test purposes covered is the first subclause of the preprocessor subclause. All general or other test purposes covered by this preprocessor test case are listed in this section.

The application objects covered by a preprocessor input specification are referenced in the Id column of the input specification. A statement to that effect is included in this subclause of the ATS.

7.3.2 Preprocessor input specification

For preprocessor test cases, the input specification is a formalized presentation of an example instantiation of the AP information requirements. See annex B for the formal grammar for the preprocessor input specification. The preprocessor input specification is represented by a combination of a table and supporting graphic and text information (sometimes called "hardcopy").

The preprocessor input specification is described using the application element terminology of the AP to which the IUT claims conformance. Additional information may be needed if the application elements do not express all that is required for an unambiguous description of the preprocessor input. Additional information to remove ambiguity in an input specification could include:

— diagrams or other graphical representations that clearly describe requirements but do not contain any more information than required to unambiguously represent the semantic content;

EXAMPLE 1 AP 203 does not specify the details of a B-Rep model in its ARM. Therefore the ATS developer may choose to provide the details of a B-Rep solid in the form of a reference to a figure showing a drawing within the input description.

EXAMPLE 2 If the AP does not support the exchange of presentation information such as color, line fonts or dimensions, no graphical representation used in an abstract test case for that AP shall suggest a specific color or line font. Likewise, any dimensional information provided shall be clearly identified as for information only.

— AIM constructs supporting and clarifying the information requirements.

The preprocessor input specification table is composed of five columns. The purpose of each column and instructions on populating the table appear in the following subclauses.

7.3.2.1 Id column

The input specification Id column serves a dual purpose: an identifier for the application object that can be referenced as the value of some other application assertion, and a reference to the AP clause number where the application object is defined. The identifier is constructed with an '@' symbol followed by the clause 4.2 subclause id that is associated with the AE object in this row of the table. If there is more than one instance of an application object in the table, the identifier for that application object shall be followed by a "." and an integer that makes the input specification Id unique across all application objects in the table.

EXAMPLE 1 An application object Application_object1 has two attributes, attribute1 and attribute2. The application object is defined in clause 4.2.10 of the relevant AP. The input specification Id's would be @10:

Id	V	Application Elements	Value	Req
@10	*	Application_object1	#20	M
	*	Application_object1.attribute1	#20, “Value of attribute1”	S
	*	Application_object1.attribute2	#30,$	S

EXAMPLE 2 If there are two instances of Application_object1 in the table, the first instance would have an Id of @10.1 and the second instance would have an Id of @10.2.

Id	V	Application Elements	Value	Req
@10.1	*	Application_object1	#20	M
		Application_object1.attribute1	#20, “Value of attribute1”	S
		Application_object1.attribute2	#30,$	S
@10.2	*	Application_object1	#200	M
	*	Application_object1.attribute1	#200,”Other value of attribute1”	C1
	*	Application_object1.attribute2	#300,”Value of attribute2”	M

7.3.2.2 V column

The V (Verdict) column identifies whether the associated application element is assigned a verdict in this test case. An asterisk '*' indicates that the application element is assigned a verdict using a derived verdict criterion according to the rules defined in annex C. An integer also indicates the application element is assigned a verdict, and is used to reference one of the specific verdict criterion (or range of verdict criteria) identified in the verdict criteria subclause of the preprocessor test case. If the column is blank this indicates the application element is not assigned a verdict in this test case.

EXAMPLE Application_object2 is a complex object for which the general and derived verdict criteria do not suffice. A reference is placed in the V column to the two verdict criteria associated with that object:

Id	V	Application Elements	Value	Req
@20	1-2	Application_object2	#3020,<see Figure 1>	S

vc1 The model realized by the IUT corresponds in shape to the model represented in Figure 1.

vc2 The radius of the central hole of the ring is 5.0 mm.

7.3.2.3 Application elements column

The Application Elements column lists the application elements within the scope of this input specification and for which sample data are provided. The following forms are used:

— Application objects are represented using the application object name.

— Categorizations are the representation of a supertype component in an instance of a subtype object. Categorizations are represented by text of the form Application_object1 (as Application_object2), where Application_object1 is the supertype and Application_object2 is the subtype. If there is more than one subtype categorization instance present in the input specification, then there is a separate row for each categorization covered.

— Application attributes are represented as the application object name followed by a "." followed by the attribute name. They are listed first under the application object. If an aggregate attribute has more than one element present there is only one row for the attribute and the value column shows the values for each of the elements enclosed in parentheses and separated with a ",". For clarity each element's value for the aggregate can appear on a separate line in the value column.

— Assertions are represented as they are in clause 4 of the application protocol followed by the role-description in parenthesis. They are listed after the attributes for the application object. Both the primary relationships and the inverse relationships are listed under the application object. The format of the relationship is: Application_object1 to Application_object2 (role) where 'role' is the role description from clause 4 of the AP. For inverse relationships, the role string is the inverse role description. If a one to many assertion has more than one relationship present then there is only one row for the assertion and the Value column is used to list each related application object Id separated by a ",".

Rows documenting all the application elements related to a particular application object are grouped together in the following order:

— application object. If the application object is instantiated as one of its subtypes, then the supertype application object component of this instance is labelled as a categorisation;

EXAMPLE 1 If an entity A is a defined supertype of entity B, and the application object being realised by the test data is the supertype A as itself, the application object name 'A' would be the entry in the application element column. If, however, the object to be realised by the test data is the subtype B, the entry for the row representing the supertype component of a complete instance of B would be 'A (as B)'.

— attributes for the application object (if any);

— assertions for the application object (if any);

— subtype application object (if any);

— attributes for the subtype application object (if any);

— assertions for the subtype application object (if any);

— continue with any other subtype application objects as necessary.

EXAMPLE 2 An application object Application_object1 has a defined categorisation: Each Application_object1 can be an Application_object2 or an Application_object3. The object also has a defined attribute and has a related application assertion.

An example simple instance representing an instance of the supertype Application_object1 as itself (that is not related to any other instance of Application_object4) appears as follows, where the definition of Application_object1 is given in clause 4.2.7 in the relevant AP:

Id	V	Application Elements	Value	Req
@7	*	Application_object1	#20	M
	*	Application_object1.attribute1	#20, “Value of attribute1”	S
	*	Application_object1 to Application_object4 (is forward related to)	<not present>	M

An example complex instance representing an instance of the subtype Application_object2 (that is also related to an instance of Application_object4) appears as follows, where Application_object2 is defined in clause 4.2.9 of the relevant AP:

Id	V	Application Elements	Value	Req
@7	*	Application_object1 (As Application_object2)	#20	M
	*	Application_object1.attribute1	#20, “Value of attribute1”	S
	*	Application_object1 to Application_object4 (is forward related to)	@40	S
@9		Application_object2	#2020	M
		Application_object2.attribute2	#2020, 99.9	S
@40		Application_object4	#400	M
		Application_object4.attribute3	#400, “Value of attribute3”	S
	*	Application_object1 to Application_object4 (is inverse related from)	@7	M

Additional levels of supertype/subtype relationship are documented in a consistent fashion. An example complex instance representing an instance of the subtype Application_object5 (defined in clause 4.2.11 in the relevant AP), a subtype of Application_object2, itself a subtype of Application_object1 appears as follows:

Id	V	Application Elements	Value	Req
@7	*	Application_object1 (As Application_object2)	#20	M
	*	Application_object1.attribute1	#20, “Value of attribute1”	S
	*	Application_object1 to Application_object4 (is forward related to)	@40	M
@9		Application_object2 (as Application_object5)	#2020	M
		Application_object2.attribute2	#2020, 99.9	S
@11		Application_object5	#555	S
	*	Application_object5.attribute5	#500, “Value of attribute5”	M

7.3.2.4 Value column

The value for the application element is listed in the Value column. There are several forms possible:

— The value column for an application object contains a link to the mirror AIM entity in the post-processor input specification by referencing the entity instance name (#number).

— The value column for categorisations follows the rule for simple application objects above.

— The value column for an attribute contains a link to the related AIM entity in the mirror postprocessor input specification by referencing the entity instance name. The link serves as a pointer for a test case realiser to identify where the mirror value is located that can be changed.

EXAMPLE 1 A preprocessor input specification for ISO 10303-303 includes a Part application object with two of its attributes defined as follows:

Id	V	Application Elements	Value	Req
@115	*	Part	#1	M
@118	*	Part.part_nomenclature	#1, “Flat ring gasket”	S
@119	*	Part.part_number	#1,”11113”	S

An excerpt from the postprocessor input specification for this test case is as follows:

```
...

#1=PRODUCT('11113',"Flat Ring Gasket','Description for part 11113',(#2));

#2=MECHANICAL_CONTEXT('detailed design',#3,'mechanical');

...
```

The mapping table for ISO 10303-203 maps Part.part_nomenclature into product.name and part_number into product.id. The Value column for Part.part_nomenclature and Part.part_number identifies the PRODUCT entity with entity instance name, "#1", in the postprocessor specification as the AIM entity instance that defines the mirror values.

— The value column for attributes that have simple values shall also include a representation of the value for the attribute. This value follows the entity instance name (see @118 and @119 in example 1). The value for an attribute is simple if it maps to a base type. Strings shall be enclosed in single quote marks. Numerical, boolean, and enumerated values shall not be enclosed in single quotes. Sets of more than one value shall be enclosed in parentheses and separated by commas (see grammar rule 72 in B.1). For purposes of clarity, the AIM entity attribute name may be followed by an "=" before the value (see rule 72 in B.1). If the attribute is not present in the input model then the keyword, "<not_present>” shall be used. If an attribute maps to an AIM entity type, then the ATS developer must determine if there is a non-ambiguous way to represent the value using one of the preceeding forms. The mirror entity instance name from the corresponding postprocessor input specification is required. Additionally an external reference to a figure or other explanatory text can also be used in the value column (see rule 72 in B.1).

— The values for aggregate attributes are enclosed in parentheses and separated by a "," (see rule 72 in B.1).

EXAMPLE 2 The following examples are legal values for the Value column of an attribute:

#2020, TRUE

#5, approved

#7,"

#6,'Flat ring gasket'

#9,1.000

#3120, (2.0, 2.5, 1.8)

(#100,2.0), (#200,3.0)

#100, #101, 'John Smith'

#10,first_name = 'John'

#10,(first_name = 'John', last_name = 'Smith')

<not_present>

#3030, <see Figure 1>

— The value column for assertions contains the Id number (@number) of the related application object. If the assertion is a one to many assertion with more than one application object relationship present, then the value column contains a list of the Id numbers separated by a ",". If the assertion is not present in the input model then the keyword "<not_present>" is used.

— Any preprocessor input specification that has more than one associated postprocessor input specification must provide an appropriate value for each postprocessor mirror. Each value is preceded by the identification number of the appropriate postprocessor specification followed by a ':'. The values are separated by blank space or a newline.

7.3.2.5 Req column

The Req column specifies the requirements on the values in the Value column. Values in the value column shall be one of mandatory, suggested, or constrained (denoted by a 'M', 'S', or 'C<n>' in the Req column). Values in an abstract test case are normally suggested values which means the test case realiser is free to change those values provided that the new value is still of the same type. Values shall be mandatory due to requirements of the standard (EXPRESS rules, mapping table requirements, or test purposes). Values shall be constrained if they are not mandatory but any changes to the value can cause non-obvious changes in other parts of the test case. All values in the preprocessor input specification shall be suggested unless one of the following rules applies. Rules for mandatory values take precedence over rules for constrained values.

a) The value for an assertion present in the input specification table is mandatory.

Rationale: Assertions describe the structure of the file. To change their value implies changing the structure of the file. Although it is conceivable that one could trace through an assertion and all its related elements and determine whether it could be changed without affecting any other verdict in the file, such a task would be extremely laborious. It would also make maintenance extremely difficult as every reassignment of verdicts or changes to the test case would force a reassessment of each assertion. Hence it is strongly recommended that all assertions be labeled as M. Note that not all asser-

tions for an application object appear in the input specification table. This rule only applies to those assertions actually listed in the table.

b) The value for an application object that has attributes is mandatory.

Rationale: This is merely a convention. The value requirement for an application object with attributes is defined by the value requirements for each of the attributes.

c) The value for an application object that has no attributes follows the same rules as attributes below.

Rationale: The Value column for an application object without any attributes defines the value for the object. Therefore the Req column shall specify the constraints (if any) on changes to the value by the test case realiser.

d) The value for an attribute that is assigned a verdict and that has a test purpose of the form, "Application_object with attribute = value" is mandatory.

Rationale: Attributes that are assigned a verdict and that have a restricted set of values or are mapped to AIM attributes of type, ENUMERATION, BOOLEAN, or LOGICAL will have test purposes identified for each of the allowed values. In order for any of those test purposes to be correctly assigned a verdict in a test case, the value assigned cannot be changed and therefore it is mandatory.

e) The value for an attribute that is assigned a verdict and that has a value restricted by a value constraint in the mapping table is mandatory.

Rationale: Attributes that are assigned a verdict and that have values restricted by a value constraint in the mapping table shall be analyzed with those values.

f) The value for an attribute that is assigned a verdict but whose value is <not_present> is mandatory.

Rationale: One of the test purposes for an optional attribute uses the form, "Application_object with attribute not present". In order to verdict this test purpose it is mandatory that the attribute not be present hence there can be no other allowed "value" for the attribute.

g) The value for an attribute is constrained when the attribute is assigned a verdict and when changes to the value might cause changes to other parts of the test case that may not be obvious.

Rationale: Marking a value as constrained serves as a warning to the test case realiser that changes to this value can cause changes to other parts of the test case (such as other parts of the input model or other verdict criteria) that may not be obvious. The constraint referenced by this column shall clearly explain what parts of the test case might be impacted by a change in the value.

h) When the semantic value defined in the table for an attribute has specific domain importance the value is constrained.

Rationale: ATS developers may determine that a certain value should be assigned to an application element since the value tests for conditions that are important to the domain of the AP but that are not necessarily explicitly identified by the standard. For example, in ISO 10303-203 the ATS developers wanted a range of different kinds of geometry to be used in the test suite to cover geometry of interest to the domain but not called out in the information requirements. If these values were left as suggested values, a test case realiser would be free to substitute a completely different set of geometry that would no longer serve the purpose of the domain expert's test suite.

The Req column specifies whether the value is mandatory, suggested or constrained according to the rules defined above. All the constraints in the table shall reference one or more notes detailed in the Constraints on values subclause of the test case. The explanatory text associated with the constraint shall clearly identify the reason for the constraint and provide sufficient description of what aspects of the test case would have to be changed if the value were changed. If the value can only be changed within certain bounds or according to certain criteria then those shall be clearly identified. For example a solid ring might be used as the test shape in ISO 10303-303. Certain dimensions of the ring might be changed but only so the ring still remains circular with a hole in the center.

7.3.2.6 Pruning the table

In order to reduce the size of the input table, entries shall be removed. Those rows where the application element is not assigned a verdict and whose value in the Value column is <not_present> shall be removed from the table.

7.3.3 Preprocessor constraints on values

This optional section is only included if any values in the input specification are constrained. This section documents the list of constraints for the preprocessor input specification. The constraint descriptions define the nature of the constraint and what changes to the value are allowed. Values shall be constrained if they are not mandatory but any changes to the value can cause non-obvious changes in other parts of the test case. Changes to the values may also be constrained by the test case to a defined set of allowable alterations.

7.3.4 Preprocessor verdict criteria

Verdict criteria are statements of the requirements that shall be satisfied for a PASS verdict to be assigned. They are assertions on the observable output of an implementation under test resulting from its execution of a test case. To verify that the requirements for a particular test purpose are fulfilled by an IUT, verdict criteria are established in the form of assertions on the IUT's output that can be evaluated to PASS (test was successfully exercised) or FAIL (test showed a conformance violation by the IUT), or INCONCLUSIVE (it was not possible to determine a PASS or FAIL verdict).

Verdict criteria are documented as either general, i.e. applicable to multiple test cases, or specific to a single test case. Verdict criteria are used in the three basic kinds of testing analysis outlined in ISO 10303-34: syntax, structure, and semantic. Accordingly, the verdict criteria shall address all three of these areas:

— Syntax analysis can be addressed by a single general verdict criterion (which can be applicable across an entire ATS) which requires that all exchange structures produced by an ISO 10303-21 preprocessor meet the syntax requirements of ISO 10303-21. Likewise, a single general verdict criterion can require an ISO 10303-22 preprocessor to meet the appropriate syntax requirements of ISO 10303-22.

— Structure analysis can be addressed in the ATS by a single general verdict criterion that requires that the structure of the AIM is correctly maintained. For the purposes of test case realisation, an algorithm for the generation of an explicit series of specific structural verdict criteria derived from the AP mapping table is detailed in C.1. Additional specific structural verdict criteria may also be developed to address specific aspects of AIM structure.

— Semantic analysis can be addressed in the ATS by a single general verdict criterion that requires that the semantics of the preprocessor input specification be correctly maintained. For the purposes of test case realisation, an approach for deriving specific semantic verdict criteria from the preprocessor input specification table is provided more detail in C.2. Additional specific semantic

verdict criteria may also be developed to help determine whether the information requirements in the input specifications are maintained.

Derivable verdict criteria are developed corresponding to both the preprocessor and postprocessor input specifications for the purposes of test case realisation. The task of generating the derivable verdict criteria is not the responsibility of the ATS developer, but is part of the construction of an executable test suite. Derivable verdict criteria shall not be documented in the abstract test suite. Only specific verdict criteria are to be documented in this section of the abstract test case. In addition, any general verdict criterion defined in clause 5 of the abstract test suite that applies to this test case shall be identified here by reference, according to the boilerplate text for this section provided in annex A.

Specific verdict criteria are not directly derivable from the AP. They verify the satisfaction of test purposes not directly relating to the reference path of the AP mapping table or mapping specification. Specific verdict criteria are often associated with specific other test purposes or semantic validation at a level of detail beyond that provided by the AP application elements. These specific semantic verdict criteria are typically associated with a general semantic test purpose.

EXAMPLE 1 The general test purpose

g5 The output of an IUT shall contain all the semantics defined by the input model.

may have the associated specific verdict criteria, where Figure 1 represents the geometry and topology of the input model.

vc1 The diameter of the largest circle in this model is 2.5 cm.

vc2 Geometry data in the postprocessor output preserves the semantics of the specifications given in Figure 1.

Each specific verdict criterion shall be explicitly associated with each test purpose it addresses. A complete abstract test suite shall have at least one verdict criterion associated with every test purpose listed in the abstract test suite.

Specific verdict criteria for preprocessor testing need to be provided for the individual abstract test cases where the preprocessor input description provides more detail than the related AP's application elements. Such specific verdict criteria are required to ensure that a conformance testing laboratory using the abstract test suite will, in fact, cover all the aspects of the input specification when analyzing the IUT's output.

EXAMPLE 2 In the case given in the example above, the verdict criteria derived from AP 203's mapping table will cover the test whether an advanced_brep_shape_representation is present at all. In order to capture the additional information provided by the drawing, the ATS may contain the following verdict criterion:

vc1 The geometry in the preprocessor output matches the specifications of the drawing given in Figure 5.

7.4 Postprocessor testing

Each abstract test case shall contain one or more postprocessor input specification with the same input semantics as the preprocessor input specification. More than one postprocessor input specification may be required to fully exercise the range of values allowed in the postprocessor implementation. This happens as a result of multiple ways in the AIM EXPRESS to encode a particular set of semantics expressed in the application elements.

EXAMPLE For the application object date, there are a number of legitimate ways to represent date in the AIM EXPRESS (e.g., calendar date, ordinal date). If the application semantic to be evaluated is date, then the various input specification variations of date shall be presented in the input specifications for postprocessors.

The postprocessor subclause(s) shall be the second and subsequent subclauses in the abstract test case. Each postprocessor subclause is divided into separate sections that include, at a minimum, test purposes covered, a reference to the postprocessor input specification, and verdict criteria.

7.4.1 Postprocessor test purposes covered

This section is used to list or reference all general and specific other test purposes covered by this postprocessor input specification.

The application objects covered by a postprocessor input specification (by virtue of the fact that it mirrors the preprocessor input specification) are referenced from the Id column of the preprocessor input specification. A statement to that effect is included in this subclause of the ATS.

7.4.2 Postprocessor input specification

For postprocessor testing the input specification is a formalized presentation of an example instantiation of the AP's AIM. The postprocessor input specification shall be in ISO 10303-21 or a form from which a ISO 10303-21 exchange structure can be generated. Accordingly, the postprocessor input specification shall be provided in one of the following:

— ISO 10303-21;

— ISO 10303-12 (EXPRESS-I).

Abstract test suite developers should choose one method for use throughout the abstract test suite for consistency and readability.

The input specification defines the concepts, constructs, and relationships to be tested without mandating specific data values, except where the test purpose itself mandates specific values. If suggested values are provided as part of an input specification, they shall not represent any actual person, company, or organization.

A normative annex in the ATS establishes the link between each ATC and its digital postprocessor input specification.

The actual postprocessor input specifications for all ATCs shall be provided in digital form via the Internet. They are referenced in the ATC.

7.4.3 Postprocessor verdict criteria

As with the preprocessor, the postprocessor verdict criteria are statements of the requirements that shall be satisfied for a PASS verdict to be assigned. The specific verdict criteria shall be documented in the test case in natural language, formalized as necessary to provide a clear, concise statement. Postprocessor verdict criteria shall be phrased as statements that can be evaluated as PASS, FAIL, or INCONCLUSIVE. In addition, any general verdict criterion defined in clause 5 of the abstract test suite that applies to this test case shall be referenced.

In the case of a postprocessor test, the input of the IUT is described using the AIM terminology of the AP to which the IUT claims conformance, i.e., a file encoded according to ISO 10303-21 or a sequence of calls according to ISO 10303-22. The output of a postprocessor is dependent on the IUT but is used in the context of an AP. The syntax and structure of the application elements of the AP are

therefore suitable to describe the IUT's output in a system-independent manner. The verdict criteria use the terminology of the application elements to allow the verification of the IUT's behavior.

For each ATC, the expected output of the postprocessor is described by the input specification of the associated preprocessor. This is presented in the form of a table giving suggested or mandatory values for application objects and their attributes. For the case of suggested values, the testing laboratory conducting the conformance test will assign fixed values during the creation of executable test cases and assign fixed values during the creation of executable test cases for postprocessor input. This is equivalent to having a table that uses exclusively mandatory values. This finalized table is referred to as the Expected Output Specification (EOS). The EOS is a precise representation of the expected output from a conforming postprocessor. Therefore the verdict criteria can be directly derived from the EOS. An approach to translate the EOS to a series of verdict criteria is presented in annex C. The task of exercising the algorithm is left to the testing laboratory as part of the executable test suite. These derived verdict criteria are not documented in the ATS.

When the expected output is described using constructs other than application elements, additional specific verdict criteria are required because the success of the translation cannot be judged by testing the presence of application elements alone. These specific verdict criteria must also be included in ATS.

EXAMPLE In the case where an AP does not specify details of a boundary representation model in its ARM and the details are provided as hardcopy in the preprocessor input specification, the verdict criteria derived from the expected output specification will only cover the test whether an advanced_b_rep is present. In order to capture the additional information provided by the drawing, the ATS may contain the following verdict criterion: "The geometry data in the postprocessor output matches the specifications of the drawing given in Figure x.".

7.5 Additional test case information

In an abstract test case, information beyond that given by the previous entries can optionally be included to ensure that an executable test case can be realized. If so, then that information shall be contained in separate sections of an abstract test case. These section can appear in any or all of three locations: in the common section at the beginning of the abstract test case following the test case summary, in the preprocessor subclause, and in the postprocessor subclause. These sections shall be the last sections of the subclause of the abstract test case in which it appears.

The kind of information that shall appear as an additional subclause includes:

— sequence of execution;

— extra details on the test.

7.5.1 Sequence of execution

In the execution of a test case, the sequence of operations to be performed can be significant. If the order of creation, use, or modification of the components of the input specification can affect the test case outcome, then additional information identifying the required sequence shall be included in this section of the abstract test case. Such information may be presented through the use of natural language or by a means such as ISO 10303-22.

Any information necessary to make clear the application of the execution sequence shall also be included.

7.5.2 Extra details on the test

If any further information is needed to describe the abstract test case, it shall be presented in a subclause called "Extra details."

8 Abstract test suite annexes

The abstract test suite shall have at least three annexes that provide information on:

— the relationship of the abstract test cases with the conformance classes of the AP;

— an ISO-required annex providing information object registration;

— the association of test case to electronic postprocessor input specification files.

The abstract test suite may optionally have an annex that provides

— usage scenarios not documented in the AP.

The abstract test suite part may include other normative and informative annexes as deemed appropriate by the test suite developers. In particular, it should include an informative annex delivered in electronic form that contains ISO 10303-21 exchange structures corresponding to the abstract test cases.

8.1 Documentation of conformance classes

The abstract test suite shall contain a normative annex that defines which abstract test cases are appropriate for which conformance classes of the AP. If the AP has no documented conformance classes, then the annex shall include a statement to that effect.

8.2 Documentation of postprocessor input specification file names

The postprocessor input specification for each test case is supplied electronically on the Internet. The abstract test suite shall contain a normative annex that lists the file names of the postprocessor input specifications supplied and associates them with the postprocessor subclause of each test case.

8.3 Documentation of usage scenarios

The abstract test suite shall contain an informative annex that documents any usage scenarios that led to the creation of postprocessor input but that were not documented in the respective application protocol.

9 Abstract test suite validation

The abstract test suite developers shall ensure that the abstract test suite is valid. Validation is required for both the abstract test suite as a whole and for each individual abstract test case. Validation is accomplished by the approach presented here. Validating the "correctness" of an abstract test suite and its constituent test cases will likely be achieved through three processes:

— verification of internal details through manual walk-through of selected samples of test cases;

— execution in real testing situations, and feedback of detected problems;

— use of reliable tools both in the generation and analysis of the tests.

9.1 Validating the abstract test suite

Abstract test suite validation involves three aspects. The first aspect of validity is established when the set of test purposes satisfies the implicit and explicit set of AP conformance requirements. The second aspect is established when the individual abstract test cases have been validated for their individual form and content. The third aspect of validity is coverage. Each abstract test case contributes to the overall coverage of the abstract test suite. The following are the criteria that apply to evaluating a test suite as a whole. A good test suite shall:

— contain test cases that adequately cover the test purposes;

— contain a set of test purposes sufficient for the application domain;

— contain all the information needed for actual testing;

— not contain unnecessary redundancies;

— be practical for actual testing.

Validating an abstract test suite against its relevant AP is necessary to ensure that the entire abstract test suite serves the needs of the AP.

9.2 Validating abstract test cases

Individual abstract test cases shall be validated against the requirements specified in clause 7 of this document. The following are the criteria for individual abstract test case validation:

— the test case identifier is present and correct;

— the test case summary is present and correct;

— the test purpose coverage is present and correct;

— the verdict criteria are present and correct;

— the input specification present is and correct.

Annex A
(normative)

AE and AIM test purposes derivation

This annex provides a syntax and specific guidance for deriving test purposes. Test purposes can be derived from both the ARM application elements and the AIM EXPRESS of an AP. While derived test purposes are not explicitly documented in the ATS, AP projects shall create these test purposes early in the AP development process and review them as a means of validating the ARM and AIM models. Test case realisers may also use the guidance in this annex to document test purposes as part of the executable test suite.

NOTE The specific documentation form for test purposes and test purpose identifiers described in this annex is not a part of the abstract test suite documentation. The formal description of the derived test purposes in this annex is to be used in ARM validation and by testing laboratories in developing executable test suites. The suggested format and identifiers are just suggestions.

A.1 Test purpose syntax

This annex specifies a method for documentation of test purposes for ISO 10303 abstract test suites. This method is applicable to the formal specification of:

— test purposes derived from the application elements (AEs) of an application protocol;

— test purposes derived from the application interpreted model (AIM) of an application protocol;

— test purposes derived from domain requirements;

— test purposes derived from an implementation method;

— test purposes derived from standards that are referenced by an application protocol.

This method is also applicable to the identification of test purposes that are derived from other sources.

The following are outside the scope of this method:

— the formal specification of general test purposes;

— the specification of test purposes for any purpose other than the development of ISO 10303 abstract test suites;

— the requirements for software tools that can be used to develop, validate or process test purposes according to the syntax defined in this document.

A comprehensive set of test purposes for the information requirements and application interpreted model is likely to be large and complex. Supplying the complete context for every test purpose will reduce the readability and clarity of the information. For this reason it is recommended that an explanation of how to interpret the AE and AIM test purposes is part of the test purpose documentation. The following is a suggested version of that text:

Each test purpose statement identifies some specific element from the AEs or the AIM. Every test purpose statement implicitly requires that the identified element, as specified in the test purpose

statement, will be correctly instantiated in the implementation under test in at least one test case within the test suite.

The following subclauses specify the lexical elements of test purposes and the grammar rules that these elements obey. The notation used to present the syntax of test purposes is defined in clause 6 of ISO 10303-11.

A.1.1 Keywords

The following rules are used to represent the keywords of test purposes. Case is not significant except within string literals. Test purposes may be written using upper, lower, or mixed case letters. However, certain keywords are defined in all capital letters and shall be written as such in the actual test purposes.

0 ae = 'ae' .

1 aim = 'aim' .

2 and = 'AND' .

3 as = 'as' .

4 atc = 'atc' .

5 element = 'element' .

6 false = 'FALSE' .

7 many = 'many' .

8 not-present = 'not present' .

9 of = 'of' .

10 one = 'one' .

11 other = 'other' .

12 true = 'TRUE' .

13 unknown = 'UNKNOWN' .

14 with = 'with' .

15 zero = 'zero' .

A.1.2 Character classes

The following rules define the various classes of characters that are used in constructing the tokens in A.1.3.

16 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .

17 digits = digit { digit } .

18 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'.

19 sign = '+' | '-'.

A.1.3 Lexical elements

The following rules specify how certain combinations of characters are interpreted as lexical elements within the language.

20 integer-literal = digits .

21 logical-literal = FALSE | TRUE | UNKNOWN .

22 real-literal = digits '.' [digits] ['e' [sign] digits] .

23 simple-id = letter { letter | digit | '_' } .

24 special-char = '!' | '"' | '#' | '$' | '%' | '&' | '+' | ',' | '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' | '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' | '{' | '|' | '}' | '~' | '(' | ')' | '*' .

25 string-literal = \q { (\q \q) | letter | digit | special-char | \s | \o } \q .

26 strict-literal = \q { letter | \s | \o } \q .

A.1.4 Grammar rules

The following rules specify how the previous lexical elements may be combined into constructs for test purposes. The primary syntax rules for test purposes are rules 30, 31, and 42.

27 aggr-spec = of \s (zero | one | many | integer-literal) \s element[s] .

28 aim-attribute-spec = attr-aggr | attr-value | attr-optional .

29 aim-test-purpose = aim test-purpose-id \s object-name \s [aim-attribute-spec | relationship-spec | categorisation-spec] .

30 ae-attribute-spec = simple-attr | attr-aggr | attr-value | attr-optional .

31 ae-test-purpose = ae test-purpose-id \s object-name \s [ae-attribute-spec | relationship-spec | categorisation-spec] .

32 attr-aggr = simple-attr \s '=' \s aggr-spec .

33 attr-optional = simple-attr not-present .

34 attr-value = simple-attr \s '=' \s attribute-value .

35 attribute-name = simple-id .

36 attribute-value = literal | enumeration-reference .

37 categorisation-spec = as simple-categorisation | complex-categorisation .

38 complex-categorisation = '(' object-name AND object-name { AND object-name } ')' .

39 enumeration-reference = simple-id .

40 literal = integer-literal | logical-literal | real-literal | string-literal .

41 object-name = simple-id .

42 other-test-purpose = other test-purpose-id string-literal .

43 relationship-spec = role-description (zero | one | many) object-name .

44 role-description = { letter | \s | \o } .

45 simple-attr = with attribute-name .

46 simple-categorisation = object-name .

47 syntax = test-purpose {test-purpose} .

48 test-purpose = ae-test-purpose | aim-test-purpose | other-test-purpose .

49 test-purpose-id = digits .

A.2 AE test purpose derivation

Application element (AE) test purposes are derived from the normative information requirements defined in an application protocol. This annex describes a process to generate an explicit, comprehensive list of AE test purposes from the information requirements of clause 4 of the associated AP.

The meaning of each test purpose derived from the information requirements is given in the following statement:

Correctly instantiate in the implementation under test the semantic associated with the unique application concept corresponding to (insert test purpose here) in at least one test case within the test suite.

The SC4 Supplementary directives – Rules for the structure and drafting of SC4 standards gives specific guidance on the documentation of these information requirements. It states that the information requirements are specified as a set of units of functionality, application objects, and application assertions. An application object is an atomic element that embodies a unique application concept and contains attributes that specify the data elements that the object is comprised of. The assertions pertain to individual application objects and to relationships between application objects. The application objects and application assertions are documented in the language of the application, but mixed with the terminology of data modeling.

AE test purposes are derived from the following aspects of the application reference model:

— application objects;

— application objects with categorisations (subtypes);

— application objects with mandatory attributes;

— application objects with aggregated attributes;

— application objects with specific attribute values;

— application objects with optional attributes;

— application assertions describing relationships between application objects.

This method results in the generation of at least one AE test purpose for each normative information requirement in clause 4 of the AP. A general term application element specifies either an application object, a specific attribute defined on an application object, or an application assertion. Every application element corresponds to at least one AE test purpose. AEs shall be documented with the same convention used in clause 4 of the application protocol. The first letter of the first word of an application object name is capitalized, no letters of the attribute names are capitalized.

A.2.1 Test purposes derived from application objects

Every application object defined in the application protocol results in at least one test purpose for that application object.

A.2.1.1 Simple application objects

Application objects are defined in the subclauses of 4.2 in an AP. A simple application object is given in the AP information requirements without any defined categorisations. Each simple application object results in one explicit test purpose corresponding to an instance of that object created as itself. Each subclause of 4.2 of the AP that defines a simple application object results in one test purpose where the test purpose consists of the keyword "ae", the test-purpose-id, and the application object name.

EXAMPLE In ISO 10303-201

4.2.30 Drawing

results in the test purpose:

ae001 Drawing

A.2.1.2 Categorisations

In the subclauses of 4.2 of the AP, an application object can define categorisations using one of the following:

Each <application object name> may be a(n) <subtype application object name 1>, or a(n) <subtype application object name 2>, ... or a(n) <subtype application object name n>.

Each <application object name> is either a(n) <subtype application object name 1>, or a(n) <subtype application object name 2>, ... or a(n) <subtype application object name n>.

Each application object that has defined categorisations of the first form also results in the simple application object test purpose described in A.2.1.1.

This second form of categorisation definition describes an abstract supertype; as such the supertype instance is never to be created as itself. Therefore, each application object that has defined categorisations of the second form does not result in the simple application object test purpose described in A.2.1.1.

Each defined categorisation of either form results in one additional test purpose associated with the supertype application object, corresponding to an instance of that object created as a supertype component of the named categorisation subtype instance. This test purpose consists of the keyword "ae",

the test-purpose-id, and the phrase supertype application object name as subtype application object name.

EXAMPLE 1 In ISO 10303-203 4.2.15 Geometric_model_representation application object has the following categorisation:

Each Geometric_model_representation may be one of the following: Advanced_B_Rep (see 4.2.2), a Faceted_B_Rep (see 4.2.14), a Non_topological_surface_and_wireframe (see 4.2.18), a Manifold_surface_with_topology (see 4.2.16), or Wireframe_with_topology (see 4.2.39).

This text results in the following test purposes:

ae001 Geometric_model_representation

ae002 Geometric_model_representation as Advanced_B_Rep

ae003 Geometric_model_representation as Faceted_B_Rep

ae004 Geometric_model_representation as Non_topological_surface_and_wireframe

ae005 Geometric_model_representation as Manifold_surface_with_topology

ae006 Geometric_model_representation as Wireframe_with_topology

EXAMPLE 2 In ISO 10303-203 4.2.24 Work_request application object has the following categorisation:

Each Work_request is either a Start_request (see 4.2.33) or a Change_request (see 4.2.6).

This text results in the following test purposes:

ae010 Work_request as Start_request

ae011 Work_request as Change_request

An application object may define more than one categorisation. If an application object has at least one categorisation that defines it as an abstract supertype, then the simple application object test purpose described in A.2.1.1 shall not be generated, regardless of any other categorisations.

If an application object has two or more categorisations, combinations of these elements can be specified in test purposes. These test purposes represent an instance created as an AND combination of two different categorisation subtypes. The specification of such complex test purposes is an option available to the developer of the abstract test suite, but is not required.

EXAMPLE 3 In ISO 10303-201 4.2.28 Draughting_annotation application object has two separate categorisations. Each Draughting_annotation is either a Annotation_element, or a Dimension, or a Draughting_callout. Each Draughting_annotation may be one of the following: Model_placed_annotation, or a Sheet_placed_annotation, or a View_placed_annotation. This results in the following mandatory test purposes:

ae020 Draughting_annotation as Annotation_element

ae021 Draughting_annotation as Dimension

ae022 Draughting_annotation as Draughting_callout

ae023 Draughting_annotation as Model_placed_annotation

ae024 Draughting_annotation as Sheet_placed_annotation

ae025 Draughting_annotation as View_placed_annotation

This can also result in the following optional test purposes:

ae026 Draughting_annotation as (Annotation_element AND Model_placed_annotation)

ae027 Draughting_annotation as (Annotation_element AND Sheet_placed_annotation)

ae028 Draughting_annotation as (Annotation_element AND View_placed_annotation)

ae029 Draughting_annotation as (Dimension AND Model_placed_annotation)

ae030 Draughting_annotation as (Dimension AND Sheet_placed_annotation)

ae031 Draughting_annotation as (Dimension AND View_placed_annotation)

ae032 Draughting_annotation as (Draughting_callout AND Model_placed_annotation)

ae033 Draughting_annotation as (Draughting_callout AND Sheet_placed_annotation)

ae034 Draughting_annotation as (Draughting_callout AND View_placed_annotation)

Where such combinations are not documented as explicit test purposes, the developers of an abstract test suite should nonetheless indicate where such combinations exist. Each of the categorisation subtypes for an application object must appear in at least one of the mandatory test purposes even if it is a combination of more than one categorisation subtype.

In the subclauses of 4.2 of the AP, an application object may define implicit categorisations using the following:

The types of <application object name> that may be specified are: <type 1>, <type 2>, ... <type n>.

Such implicit categorisations are found where an application object is defined as having several values or types, where these are not themselves stated explicitly as separate application objects or as attribute values. These can result in test purposes that are equivalent to those for explicit categorisations. Like the complex combination test purposes, the specification of such test purposes is an option available to the developer of the abstract test suite, but is not required.

EXAMPLE 4 ISO 10303-201 includes an application object 2D_geometric_element, that is defined (in normative text) to be a cartesian_point, a point_on_curve, a line, a polyline, etc. ISO 10303-301 includes the test purposes:

ae023 2D_geometric_element as cartesian_point

ae024 2D_geometric_element as point_on_curve

ae025 2D_geometric_element as straight_line

ae026 2D_geometric_element as composite_curve

ae027 2D_geometric_element as bspline_curve

ae028 2D_geometric_element as offset_curve

ae029 2D_geometric_element as polyline

ae001 2D_geometric_element as circle

ae030 2D_geometric_element as ellipse

ae031 2D_geometric_element as trimmed_conic

NOTE Application elements are documented with the same convention used in the application protocol. The first letter of the first word of an application object name is capitalized. The implicit categorisations are documented with lower case to distinguish them from application objects.

In the subclauses of 4.2 of the AP, application objects that are a subtype of another application object have a first sentence with the following format:

A(n) <application object name> is a type of <supertype application object name> that is <rest of definition>.

No additional test purpose is derived from this sentence in the AP, since an appropriate test purpose is derived from the categorisation definition provided for the supertype application object.

A.2.2 Test purposes derived from application object attributes

All attributes associated with an application object result in additional test purposes associated with the application object test purpose. There will be at least one test purpose for each attribute of each application object in the AP. Each attribute of an application object is listed in the application object definition in the subclause of 4.2 of the AP in a sentence of the form:

The data associated with a(n) <application object name> are the following:

— <attribute name>.

Each attribute is defined in a subclause under an application object definition except in the case where there is only one attribute. When there is only one attribute, the attribute definition follows the above sentence in the application object definition subclause. Attribute test purposes are generated according to the following specification.

A.2.2.1 Test purposes derived from mandatory simple type attributes

In 4.2 of an AP, attribute definitions are provided in the form:

The <attribute name> specifies <definition>.

A mandatory simple attribute has no additional definitions. Every mandatory attribute of an application object defined in the application protocol results in one additional test purpose for that attribute.

EXAMPLE In ISO 10303-201, the text specifying 4.2.30.2 Drawing_number

The Drawing_number specifies the identification of a particular drawing by an organization.

results in the following additional test purpose:

ae032 Drawing with Drawing_number

A.2.2.2 Aggregate type attributes

In 4.2 of an AP, aggregated attribute definitions are indicated by an additional sentence of the form:

There may be more than one <attribute name> for a <application object name>.

Every aggregate attribute of an application object defined in the application protocol results in at least one additional test purpose for that attribute.

An aggregation results in one test purpose for the attribute with one member of the aggregation, and one test purpose for the upper limit of the aggregation.

EXAMPLE In ISO 10303-201, the aggregated attribute Contract_reference with one or many members. ISO 10303-301 results in the test purposes:

ae036 Drawing with Contract_reference of one element

ae037 Drawing with Contract_reference of many elements

Contract_reference is also defined as an optional attribute (see A.2.2.4). Therefore another test purpose results:

ae038 Drawing with Contract_reference not present

An aggregation with many members, defined as having a lower bound greater than one, results in a single test purpose.

If an explicit upper limit is specified for the aggregation then the AE test purpose still uses the words, "of many elements". If necessary a domain test purpose may specify a specific number of elements. An aggregation with a fixed number of members results in a single test purpose with the words "of <n> elements" where <n> is replaced by the fixed number of members.

A.2.2.3 Specific attribute values

Every application object that has one or more required values for an attribute results in one or more additional test purposes. One test purpose is specified for each discrete value of the attribute identified in the application protocol.

EXAMPLE 1 In ISO 10303-203, 4.2.20 Part_version has the attribute 4.2.20.3 Release_status is defined to always be one of two values: released or unreleased. This results in the following test purposes:

ae040 Part_version with release_status = released

ae041 Part_version with release_status = unreleased

An attribute of an application object that has an explicit map to an AIM entity attributes of type BOOLEAN, LOGICAL, and ENUMERATION results in one test purpose for each possible value.

EXAMPLE 2 An application protocol includes an application object B with an attribute X, that maps to an AIM attribute, whose value is BOOLEAN. The abstract test suite contains the following test purposes:

ae042/atc094 B with X = TRUE

ae043/atc094 B with X = FALSE

NOTE Attributes that reference a closed range of integer, character or string values can result in one test purpose for each possible value. The AP/ATS development team shall determine the cases where an attribute has a set of required values rather than a range within a bounded or unbounded type. It is not intended, for example, that an attribute defined as having a value between 0 and 1000 have a test purpose for each intermediate value.

It is suggested that such test purposes arise only where each value of the attribute gives a different meaning to the application object.

A.2.2.4 Optional attributes

Optional attributes are indicated in 4.2 of an AP by a sentence of the form:

The <attribute name> need not be specified for a particular <application object name>.

Every optional attribute of an application object defined in the application protocol results in one additional test purpose for that attribute, corresponding to the case of option not present. Each attribute type described in the sections above can also be an optional attribute.

EXAMPLE 1 In ISO 10303-203, 4.2.25 Planned_sequence_effectivity has an optional attribute 4.2.25.4 thru_effectivity_id. This results in the test purposes:

ae044 Planned_sequence_effectivity with thru_effectivity_id

ae045 Planned_sequence_effectivity with thru_effectivity_id not present

Aggregate attributes will generate test purposes corresponding to the bounds of the aggregate as described in section A.2.2.2 above.

Attributes having specific values result in a test purpose corresponding to each discrete value of the aggregate as described in section A.2.2.3 above.

EXAMPLE 2 An application protocol includes an application object B with an optional attribute X, that maps to an AIM attribute of type BOOLEAN. The abstract test suite contains the following test purposes:

ae046 B with X = TRUE

ae047 B with X = FALSE

ae048 B with X not present

A.2.3 Test purposes derived from application assertions

Each relationship between a pair of application objects is documented once in a subclause of 4.3 of the AP. The primary relationship is listed first. Test purposes derived from the primary relationship follow the test purposes for the attributes of the primary application object. Inverse relationships are stated in a second sentence of the application assertion subclause. Test purposes derived from the inverse relationship follow the test purposes for the attributes of the secondary application object. Assertions are written in the form:

Each <application object name> <role description> (zero, one, or many | one or more | exactly one) <application object name> [objects].

The relationship is described with a cardinality of zero, one, and/or many.

— Each application assertion having cardinality zero results in a single test purpose stating the assertion name modified with the words "<role description> zero".

— Each application assertion having cardinality one results in a single test purpose stating the assertion name modified with the phrase "<role description> one".

— Each application assertion having cardinality many results in a single test purpose stating the assertion name modified with the phrase "<role description> many".

EXAMPLE 1 In ISO 10303-203, 4.3.11 Part to Alternate_part is defined as

Each Part has an alternate of zero, one, or many Alternate_part objects. Each Alternate_part is an alternate for one or many Part objects.

This results in the test purposes for the application object part:

ae049 Part has an alternate of zero Alternate_part

ae050 Part has an alternate of one Alternate_part

ae051 Part has an alternate of many Alternate_part objects

and the test purposes for the application object Alternate_part

ae052 Alternate_part is an alternate for one Part

ae053 Alternate_part is an alternate for many Part objects

There can be more than one relationship between a pair of application objects. In this case, the subclause of 4.3 contains two pairs of assertions.

EXAMPLE 2 In ISO 10303-201, 4.3.26 Dimension to Dimension_sequence_pair is defined as

Each Dimension is the predecessor for zero, one, or many Dimension_sequence_pair objects. Each Dimension_sequence_pair has as a predecessor exactly one Dimension. Each Dimension is the successor of zero, one, or many Dimension_sequence_pair objects. Each Dimension_sequence_pair has as a successor exactly one Dimension.

The following are the resulting test purposes for the application object Dimension:

ae054 Dimension is the predecessor for zero Dimension_sequence_pair

ae055 Dimension is the predecessor for one Dimension_sequence_pair

ae056 Dimension is the predecessor for many Dimension_sequence_pair objects

ae057 Dimension is the successor for zero Dimension_sequence_pair

ae058 Dimension is the successor for one Dimension_sequence_pair

ae059 Dimension is the successor for many Dimension_sequence_pair objects

The following are the resulting test purposes for the application object Dimension_sequence_pair:

ae060 Dimension_sequence_pair has as a predecessor one Dimension

ae061 Dimension_sequence_pair has as a successor one Dimension

A.3 AIM test purpose derivation

Application interpreted model (AIM) test purposes are derived from the normative AIM EXPRESS schema defined in an application protocol. This annex describes a process to generate an explicit, comprehensive list of AE test purposes from the EXPRESS schema contained in annex A of the associated AP.

The meaning of each test purpose derived from the AIM EXPRESS schema is given in the following statement:

Correctly instantiate in the implementation under test the AIM elements associated with the unique AIM entity corresponding to (insert test purpose here) in at least one test case within the test suite.

This annex provides the details of how to generate the test purposes from the EXPRESS schema of an AP's application interpreted model (AIM).

A.3.1 Test purposes derived from entities

Every entity is associated with at least one explicit AIM test purpose. This entity test purpose implies all required features of the entity are present and correct in the instantiation. This includes all mandatory simple type attributes, all mandatory entity type attributes, all derive and inverse attributes, all local constraints (UNIQUE and WHERE), and all applicable rules.

A.3.1.1 Simple entities

Each entity defined in the AIM EXPRESS results in one explicit test purpose. This test purpose corresponds to an instance of that entity as defined itself, not as a defined subtype (see A.3.2.2).

EXAMPLE The entity

```
ENTITY simple_entity

  attr_1 : INTEGER;

  attr_2 : other_entity;

UNIQUE

  UR1 : attr_1;

WHERE

  WR1 : where_expression;

END_ENTITY;
```

has the test purpose:

aim001 simple_entity

A.3.1.2 Entities with subtypes

Subtypes of an entity defined using the ONEOF relationship, and subtype instances resulting from a ONEOF instantiation of an ANDOR relationship, do not require a test purpose. The relevant test purposes for these subtypes are generated from the EXPRESS entity definitions of the corresponding subtypes within the AIM schema of interest.

Each entity having more than one subtype defined using the AND or the ANDOR relationship has subtypes that can be combined. If an entity is defined as having subtypes that can be combined, then the complex instantiation of specific subtype combinations can result in test purposes. The specification of such test purposes is an option available to the developer of the abstract test suite, but is not required.

EXAMPLE The entity

```
ENTITY entity_with_subtypes

  SUPERTYPE OF (subtype_1, subtype_2);

  attr_1 : INTEGER;

END_ENTITY;
```

may have the specific test purpose:

aim002 entity_with_subtypes as (subtype_1 AND subtype_2)

A.3.2 Test purposes derived from attributes of entities

All attributes types referenced in this clause result in additional test purposes associated with the entity test purpose. Attribute test purposes are associated with every entity test purpose derived from an entity having attributes. These test purposes are generated according to the following specification.

A.3.2.1 Entities with aggregation type attributes

Each Aggregate type attribute in an entity results in the definition of at least one explicit test purpose associated with that attribute.

An aggregation with zero or one members (defined with bounds [0:1]) results in one test purpose for the attribute with zero members of the aggregation, and one test purpose for the attribute with one member of the aggregation, i.e. the upper limit.

EXAMPLE 1 The entity

```
ENTITY entity_with_aggr;

  array_attr : ARRAY [0:1] of base_type;

END_ENTITY;
```

has the following test purposes:

aim003 entity_with_aggr with array_attr of zero elements

aim004 entity_with_aggr with array_attr of one element

An aggregation of zero, one or many elements results in one test purpose for the attribute with zero elements of the aggregation, one test purpose for the attribute with one element of the aggregation, and one test purpose for the upper limit of the aggregation.

EXAMPLE 2 The entity

```
ENTITY entity_with_aggr;
```

```
    set_attr : SET [0:?] of base_type;
  END_ENTITY;
```

has the following test purposes:

aim005 entity_with_aggr with set_attr of zero elements

aim006 entity_with_aggr with set_attr of one element

aim007 entity_with_aggr with set_attr of many elements

An aggregation with one or many members results in one test purpose for the attribute with one member of the aggregation, and one test purpose for the upper limit of the aggregation.

EXAMPLE 3 The entity

```
  ENTITY entity_with_aggr;
    list_attr : LIST [1:10] of base_type;
  END_ENTITY;
```

has the following test purposes:

aim008 entity_with_aggr with list_attr of one element

aim009 entity_with_aggr with list_attr of many elements

An aggregation with many members results in one test purpose for the upper limit of the aggregation.

EXAMPLE 4 The entity

```
  ENTITY entity_with_aggr;
    bag_attr : BAG [2:?] of base_type;
  END_ENTITY;
```

has the following test purpose:

aim010 entity_with_aggr with bag_attr of many elements

A.3.2.2 Entities with BOOLEAN type attributes

Each attribute of type BOOLEAN in an entity results in the definition of two additional test purposes, associated with TRUE and FALSE values assigned to the attribute.

EXAMPLE The entity

```
  ENTITY entity_with_boolean;
    boolean_attr : BOOLEAN;
    attr_2 : INTEGER;
```

```
    attr_3 : REAL;
  END_ENTITY;
```

has two test purposes derived from the boolean attribute boolean_attr:

aim011 entity_with_boolean with boolean_attr = TRUE

aim012 entity_with_boolean with boolean_attr = FALSE

A.3.2.3 Entities with LOGICAL type attributes

Each attribute of type LOGICAL in an entity results in the definition of three additional test purposes, associated with TRUE, FALSE and UNKNOWN values assigned to the attribute.

EXAMPLE The entity

```
  ENTITY entity_with_logical;
    logical_attr : LOGICAL;
    attr_2 : INTEGER;
    attr_3 : REAL;
  END_ENTITY;
```

has three test purposes derived from the logical attribute logical_attr:

aim013 entity_with_logical with logical_attr = TRUE

aim014 entity_with_logical with logical_attr = FALSE

aim015 entity_with_logical with logical_attr = UNKNOWN

A.3.2.4 Entities with ENUMERATION type attributes

Each attribute of an ENUMERATION type in an entity results in the definition of one additional test purpose for each value in the enumeration list. Typically, the number of possible specific attribute values is finite and small. However, there can be a large number of enumerated values. The AP/ATS development team will have to determine the cases where an enumerated value carries a specific meaning in the context of the application protocol.

EXAMPLE Given

```
  TYPE e_type = ENUMERATION OF (e1, e2, e3, e4)
  END_TYPE;
```

the entity

```
  ENTITY entity_with_enumeration;
    enum_attr : e_type;
    attr_2 : INTEGER;
```

```
    attr_3 : REAL;
  END_ENTITY;
```

has four test purposes derived from the enumeration attribute enum_attr:

```
aim016 entity_with_enumeration with enum_attr = e1
aim017 entity_with_enumeration with enum_attr = e2
aim018 entity_with_enumeration with enum_attr = e3
aim019 entity_with_enumeration with enum_attr = e4
```

A.3.2.5 Entities with SELECT type attributes

Each attribute of a SELECT type in an entity results in the definition of one additional test purpose for each type specified in the select list. Since the number of possible specific attribute value types is finite and typically small, a heuristic of testing all possible value types has been adopted.

EXAMPLE Given

```
  TYPE s_type = SELECT (s1, s2)
  END_TYPE;
TYPE s2_type = SELECT (s3, s4)
  END_TYPE;
```

the entity

```
  ENTITY entity_with_select;
    select_attr : s_type;
    attr_2 : INTEGER;
    attr_3 : REAL;
  END_ENTITY;
```

has three test purposes derived from the select attribute select_attr:

```
aim020 entity_with_select with select_attr as s1
aim021 entity_with_select with select_attr as s3
aim022 entity_with_select with select_attr as s4
```

A.3.2.6 Entities with OPTIONAL attributes

Each OPTIONAL attribute in an entity results in the definition of at least two additional test purposes associated with the presence or absence of the attribute.

In all cases, an optional attribute contains one test purpose corresponding to the option not present. At least one other test purpose will correspond to the case of option present. Note that each attribute type described in the sections above can also be an optional attribute. As described above, depending upon the type of the attribute, more than one test purpose may be required for the option present case. Attributes that reference a simple type will contain a single test purpose representing the case of present.

EXAMPLE 1 The entity

```
ENTITY entity_with_optional;

  opt_attr : OPTIONAL INTEGER;

  attr_2 : INTEGER;

  attr_3 : REAL;

END_ENTITY;
```

has two test purposes derived from the optional attribute opt_attr:

aim023 entity_with_optional with opt_attr not present

aim024 entity_with_optional with opt_attr

Aggregate attributes will contain result in a test purpose corresponding to the bounds of the aggregate as described in section A.3.2.2 above.

EXAMPLE 2 The entity:

```
ENTITY entity_with_optional;

  opt_attr : OPTIONAL SET [0:?] of base_type;

  attr_2 : INTEGER;

  attr_3 : REAL;

END_ENTITY;
```

has four test purposes derived from the optional attribute opt_attr:

aim025 entity_with_optional with opt_attr not present

aim026 entity_with_optional with opt_attr of zero elements

aim027 entity_with_optional with opt_attr of one element

aim028 entity_with_optional with opt_attr of many elements

Attributes having specific values will contain one present test purpose corresponding to each discrete value of the aggregate as described in A.3.2.3, A.3.2.4, and A.3.2.

EXAMPLE 3 The entity:

```
ENTITY entity_with_opt_boolean;

  boolean_attr : OPTIONAL BOOLEAN;
```

```
    attr_2 : INTEGER;

    attr_3 : REAL;

  END_ENTITY;
```

has three test purposes derived from the optional attribute boolean_attr:

aim029 entity_with_opt_boolean with boolean_attr not present

aim030 entity_with_opt_boolean with boolean_attr = TRUE

aim031 entity_with_opt_boolean with boolean_attr = FALSE

A.3.3 CONSTANTS, TYPES, FUNCTIONS, PROCEDURES, and RULES

EXPRESS CONSTANTS, TYPES, FUNCTIONS, PROCEDURES, and RULES have of themselves no corresponding test purposes.

A.3.3.1 Removal of AIM test purposes

EXPRESS RULES (global constraints), FUNCTIONS, and PROCEDURES that act as local constraints, and INVERSE attributes may disallow certain AIM structure. This may disallow the instantiation of entities of a particular type, affect the cardinality of an aggregate attribute, restrict the range of values that an ENUMERATION, LOGICAL, or BOOLEAN type attributes may assume, or constrain the instantiated model in some other way. Any test purpose generated by the process described in A.3.2 and A.3.3 below that indicates an AIM structure in violation of any defined EXPRESS constraint shall not be included in the resultant set of AIM derived test purposes.

EXAMPLE In ISO 10303-203, 5.2.5.74 subtype_mandatory_representation specifies a EXPRESS RULE that requires that each instance of representation will be a shape_representation. This global constraint causes the test purpose

aim123 representation

not to be included in the final list of test purposes for ISO 10303-203.

Annex B
(normative)

Preprocessor input specification syntax

The following productions define the lexical elements and grammar rules for each of the elements used in the preprocessor input specification table. The notation builds on the lexical elements and grammar rules defined in annex A. The primary rule for the preprocessor input specification contents is rule number 68. The primary rule for the Id column is rule number 66. The primary rule for the V column is rule number 72. The primary rule for the Application Elements column is rule number 56. The primary rule for the Value column is rule number 60. The primary rule for the Req column is rule number 70.

50. ae-aggregated-value = '(' ae-value-form ')' { ',' [\n] '(' ae-value-forms ')' } .

51. ae-enumerated-value = simple-id .

52. ae-explicit-value = ae-string-value | logical-literal | real-literal | ae-enumerated-value | special-value | ae-named-value .

53. ae-explicit-value-set = '(' ae-explicit-value { ',' ae-explicit-value } ')' .

54. ae-named-value = simple-id '=' ae-explicit-value

55. ae-string-value = string-literal .

56. ae-spec = object-name | ao-attribute | appl-assertion | ao-categorisation .

57. ae-value-form1 = post-reference | ae-explicit-value | ae-explicit-value-set .

58. ae-value-form2 = post-reference ', ' (ae-explicit-value | ae-explicit-value-set) .

59. ae-value-form = ae-value-form1 | ae-value-form2 .

60. ae-value-spec = [ae-value-form | ae-aggregated-value | ao-reference] .

61. ao-attribute = object-name '.' attribute-name .

62. ao-categorisation = object-name '(as ' object-name ')' .

63. ao-reference = id-spec [{ ',' id-spec }] .

64. ap-subclause-id = digits .

65. appl-assertion = object-name ' to ' object-name '(' role-description ')' .

66. id-spec = '@' ap-subclause-id ['.' digits] .

67. post-reference = '#' digits [{ ', ' '#' digits }] .

68. preprocessor-input-spec = id-spec verdict-spec ae-spec ae-value-spec req-spec .

69. range = digits '-' digits .

70. req-spec = ['M' | 'S' | 'C' digits] .

71. special-value = '<not_present>' | '<see' ('Figure' | 'Table') real-literal '>' .

72. verdict-spec = ['*' | (digits | range) { ',' (digits | range) }] .

Annex C
(normative)

Verdict criteria derivation

Specific semantic and structure verdict criteria corresponding to the preprocessor and postprocessor input specifications may be derived for the purposes of test case realisation. The task of generating the derivable verdict criteria is not the responsibility of the ATS developer, but is part of the construction of an executable test suite. An algorithm for the generation of an explicit series of specific structural verdict criteria derived from the AP mapping table is detailed in C.1. An approach for deriving specific semantic verdict criteria from the preprocessor input specification table is provided more detail in C.2.

NOTE This annex refers to entries in an APs mapping table. Equivalent information is provided in mapping specifications, so this annex also pertains to APs containing a mapping specification in place of a mapping table. Because nearly all APs in existence contain mapping tables, the mapping table terminology was left in this annex.

These algorithms reference derived AE and AIM test purposes, also the responsibility of the test case realiser (see annex A).

C.1 Deriving preprocessor verdict criteria from the AP mapping table

The preprocessor input specification contains three different types of entries:

— entries that specify that an application object's instance is presented to the preprocessor as input;

— entries that specify values for an application object's attribute;

— entries that specify links between two application object instances (assertions).

To derive the AIM verdict criteria for these kinds of entries, the corresponding mapping table entries need to be identified. For each entry in the preprocessor input specification, the corresponding entry in the AP's mapping table is identified first. The verdict criteria are then derived from this mapping table entry.

C.1.1 Verdict criteria corresponding to application objects

The first kind of entries corresponds to the capitalized entries in the mapping table. In column 2 of the mapping table, each of these entries gives an AIM element that we call the "corresponding AIM instance" of the application element. The set of all preprocessor input specification entries of this kind therefore selects a set of corresponding AIM instances within the set of all AIM instances in the expected output of the preprocessor. Whenever a verdict criterion is related to such an AIM instance, this is expressed by adding the phrase "that corresponds to @<n>" to the phrase referencing the AIM instance, where @<n> is the preprocessor input specification entry of the application element.

Column 2 of the mapping table describes the mapping of an application object by stating the corresponding AIM element. This results in the following verdict criterion:

— An instance of <AIM element> that corresponds to @<n> is present.

Column 2 may also contain a set of alternatives in the form

```
(aim element 1)

(aim element 2)

...

(aim element n)
```

These alternatives are expressed in the following verdict criterion:

— An instance of <aim element 1> or <aim element 2> ... or <aim element n> that corresponds to @<n> are present.

NOTE The phrase "instance of x" in the context of these verdict criteria is to be interpreted as "instance of x or one of its subtypes".

Column 5 of the mapping table may contain a reference path for the application object. This results in additional derivable verdict criteria that are presented after the verdict criterion given above in an indented style. The details of deriving these verdict criteria are given in C.1.3.

In the preprocessor input specification table, additional entries may follow the entry that relates to the application object instance that specify attribute values or links to other application objects. The details how to construct verdict criteria for these are given in the following subsections. All these additional verdict criteria are to be indented relative to the first verdict criterion related to the application object as given above, in order to denote that they all relate to the corresponding AIM instance identified when evaluating the first verdict criterion.

C.1.2 Verdict criteria for attributes of application objects

In the case of an application object's attribute, column 2 of the mapping table specifies the attribute or AIM entity where the application object's attribute value is to be found, and column 5 specifies an optional reference path that establishes a link from the corresponding AIM instance of the application object to the item given in column 2. Therefore, if a reference path is given in column 5, the verdict criteria derivable from it are to be presented first according to the method described in C.1.3. One additional verdict criterion is then presented in both cases to ensure that the application object's attribute value was encoded properly.

If column 2 contains an entry of the form "`aim_element.attribute`", the basic phrase for the verdict criterion will be:

— the attribute `attribute` of the instance of `aim_element` [that corresponds to @<n>] has the value @<v>.

NOTE The phrase within brackets is only required if the AIM instance belongs to the set of corresponding AIM instances described in C.2. The phrase compares the given attribute value to the value specified in either the suggested values or the mandatory values column of the preprocessor input specification. During the creation of an executable test case, the testing laboratory will assign a fixed value for the table entry @<v> which then is promoted to the verdict criterion.

If column 2 contains an entry of the form "`aim_element`", the basic phrase will be:

— the instance of `aim_element` [that corresponds to @<n>] encodes the value @<v>.

The actual verdict criterion is constructed from the above phrase as follows:

— if column 2 of the mapping table contains a single entry, it is: "*phrase*";

— if column 2 of the mapping table contains several alternatives in the form

```
(alternative 1)

(alternative 2)

...

(alternative n)
```

thereby indicating that an attribute can be mapped in different ways, the verdict criterion is*: "phrase for alternative1, or phrase for alternative 2, ... or phrase for alternative n*".

C.1.3 Verdict criteria corresponding to application assertions

In the case of an application assertion, two base cases are to be distinguished:

— Identical mapping: Column 2 of the mapping table contains the keyword `IDENTICAL MAPPING`, which means that the two application objects involved in the assertion are represented by a single AIM entity. In this case no extra verdict criteria are required besides those that were created for the two application objects according to C.1.1;

— Path mapping: Column 2 of the mapping table contains the keyword `PATH`. This means that Column 5 of the mapping table describes the path through the instance graph from the AIM instance corresponding to the relating application object to the AIM instance corresponding to the related application object. A series of verdict criteria is derived to ensure that the whole path was constructed properly by the preprocessor. In the cases described in C.1.1 and C.1.2 where column 5 was not empty, verdict criteria are derived in the same fashion.

The following describes an algorithm for AIM verdict criteria generation from mapping paths. A path in the mapping table may contain three kinds of operators that imply forks in the path:

— OR (denoted by the syntax "`(alternative 1)(alternative 2)`");

— AND (denoted by the syntax "`[requirement 1][requirement 2]`");

— mapping rules (denoted by the syntax "`{rule-requirement}`").

In order to make verdict criteria derivation easier, first eliminate the OR-operators:

Algorithm step 1: OR elimination.

Generate, for each OR alternative, a copy of the path where one of the branches replaces the OR-construct. If the mapping path contains several points where an OR operator introduces a fork in the path, do the following:

— if the mapping table makes use of the "#n"-syntax to clarify which branches belong together, use this information for generation of the path copies;

— if it does not, repeat the OR-elimination with the copies until all OR-operators are deleted. This has the consequence that all possible combinations of the branches are generated.

Subsequent steps of the algorithm are applied to the path copies separately. The basic verdict criterion has the following general format:

Does the following set of conditions apply and are satisfied?

verdict criteria derived from copy 1

Or does the following set of conditions apply and are satisfied?

verdict criteria derived from copy 2

Or is it the case that the following set of conditions is satisfied?

verdict criteria derived from copy n.

The verdict criteria derived from the copies are indented relatively to the basic verdict criterion to clarify their relation to it.

The mapping path describes the relations between entities in a very explicit manner. SELECT trees are described in a verbose manner, and instead of using the inheritance mechanisms of `EXPRESS`, supertype/subtype relations have to be stated explicitly (i.e., the mapping table cannot directly relate to an inherited attribute `attr` of an entity `entity`, but has to state the equivalent of "entity is a subtype of super_entity, and super_entity's attribute attr is to be used such and such").

In order to avoid the generation of redundant verdict criteria, the following preprocessing steps are applied to the path:

Algorithm step 2: Elimination of select paths.

Delete all mapping table lines of the form "`select_type = selected_item`". After having deleted a consecutive set of lines, analyse the remaining lines above and below as follows:

— if the line above contains a SELECT type, propagate the entity or type from the line below to this line;

— if the line below contains a SELECT type, propagate the entity or type from the line above to this line.

Algorithm step 3: Elimination of supertype/subtype statements.

Identify lines in the mapping table that contain "<=" (is subtype of) or "=>" (is supertype of) statements. For each occurrence, identify the set of entities that is linked by a chain of "<=" or "=>" statements (mapping rules do not interrupt a chain, but can add entities to the set. At AND operators, a chain can have forks to or from the branches of the AND operator). From this set, iteratively delete the entities that are supertypes of all remaining other entities.

Note that usually a single entity will remain in the set that is required to be instantiated. If more than one entity remains, this means that a complex instance is required to be instantiated.

Replace all occurrences of entities from the original set within the chain and mapping rules included in the chain by the remaining entity set. Apply this also to the lines above the chain's start line (or start lines, in the case of AND-operators) and the lines below the chain's end line(s), as long as one of the entities from the original set is present in these neighbouring lines.

Eliminate all lines with "<=" or "=>" operators, and eliminate those mapping rules in the chain that do not contain references to attributes.

For AND operators, delete all lines from the branches that became identical for all branches during the process. If the remaining lines of the branches are identical for all (or if no lines remain), chose one branch and delete the whole AND construct. If they differ, eliminate the AND operators.

After these steps, the verdict criteria can be derived according to the following table.

Table C.1 – Verdict criteria for assertions

Mapping table entry	Resulting verdict criterion
`Entity`	An instance of *entity* is present.
`Entity.attrib -> other_entity`	The attribute *attrib* of the instance of *entity* references an instance of *other_entity*.
`Entity.attrib[i] -> other_entity`	The attribute *attrib* of the instance of *entity* references an instance of *other_entity*.
`Entity.attrib[n] ->other_entity`	The attribute *attrib*'s *n*-th component of the instance of *entity* references an instance of *other_entity*.
`Entity <- other_entity.attrib`	The instance of *entity* is referenced by an instance of *other_entity* through attribute *attrib*.
`Entity <- other_entity.attrib[i]`	The instance of *entity* is referenced by an instance of *other_entity* through attribute *attrib*.
`Entity <- other_entity.attrib[n]`	The instance of *entity* is referenced by an instance of *other_entity* through attribute *attrib*'s *n*-th component.
`Entity.attrib = val`	The attribute *attrib* of the instance of *entity* has the value *val*.
`Entity.attrib[i] = val`	The attribute *attrib* of the instance of *entity* has the value *val* for one of its components.
`Entity.attrib[n] = val`	The attribute *attrib* of the instance of *entity* has the value *val* for its *n*-th component.

NOTE 1 Phrases of the form "instance of entity" are always to be enhanced by "that corresponds to @<n>" where appropriate.

NOTE 2 Where the supertype/subtype elimination process replaced an entity by a set of more than one entity, the phrase "instance of entity" is to be replaced by "complex instance containing instances of *set-element 1, ..., and set-element n*"

NOTE 3 Apply this table line by line. If a mapping rules is encountered, and the line before the mapping rules ends in "->" or "<-", first generate the verdict criterion for the pair of lines before and after the mapping rules, then for the lines within the mapping rule.

NOTE 4 If a verdict criterion is constructed from two lines that are related by "<-" or "->" operators, do not re-use either of them for the derivation of another verdict criterion unless both lines contain "->" or "<-" operators.

C.2 Deriving postprocessor verdict criteria from the expected output specification

The expected output specification is the realisation of the preprocessor input specification for a given test case. The expected output specification (EOS) basically contains three different kinds of entries:

— entries that specify that an application object's instance is expected;

— entries that give specific values for an application object's attribute that are expected;

— entries that specify that links between two application object instances (assertions) are expected.

Verdict criteria are derived from the EOS for those entries that are marked by an asterisk in the V column in order to denote that they correspond to the AIM test purposes identified for the abstract test case.

The following table lists the constructs encountered in the EOS and gives the corresponding verdict criteria.

Table C.2 – Verdict criteria for EOS

Case	Syntax	Value	Verdict criterion
Object	@n application_element		A construct is present that corresponds to @n.
Attribute	@(n) ae.attrib	val	The equivalent of @n carries the *attrib* as *val*?
Attribute	@(n) ae.attrib[n]	val	Does the equivalent of @n carry the *attrib* with one component as *val*?
Attribute	@(n) ae.attrib	@m	Does the equivalent of @n have a construct corresponding to @m in the role of *attrib*?
Attribute	@(n) ae.attrib[n]	@m	Does the equivalent of @n have a construct corresponding to @m in the role of one *attrib*?
Assertion	@(n) ae to ae	@m	Is the equivalent of @n related to a construct corresponding to @m according to the reference path?

Annex D
(informative)

Overview of ISO 10303 conformance testing

Testing an implementation of an ISO 10303 application protocol requires assessing capability in three areas: syntax, structure, and semantics. Syntax analysis involves checking that the output exchange structure of a preprocessor satisfies all the requirements of the applicable implementation method(s). Structure analysis ensures that the data model represented in a preprocessor output exchange structure satisfies all structural requirements of the AIM EXPRESS long form of that AP. Semantic analysis verifies that the semantics of the information requirements supported by the application protocol of interest are accurately conveyed in the observed output of both preprocessors and postprocessors. Numerical analysis is a part of semantic analysis that involves checking specific numeric output values against the corresponding input data values for equality within a specified numerical accuracy for testing.

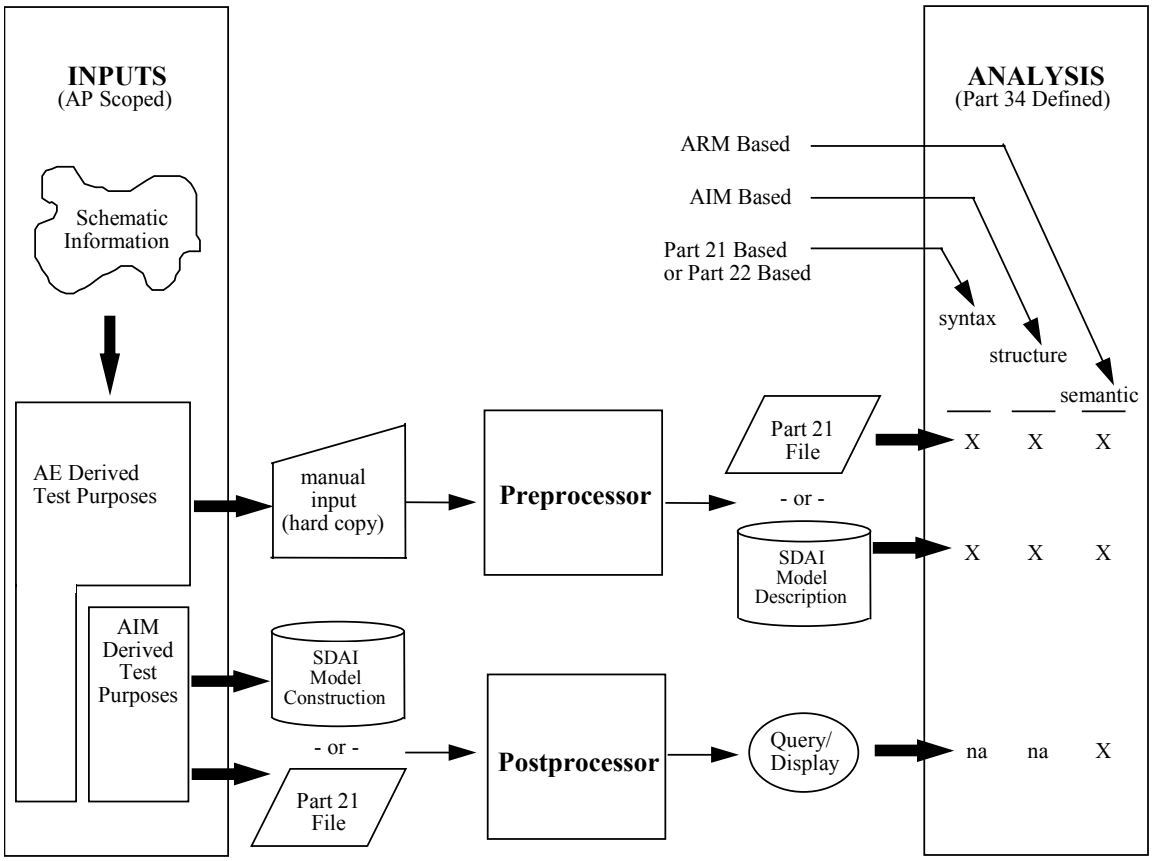

Figure D.1 – Conformance test process

Implementations of an ISO 10303 application protocol are of two forms: preprocessors (export exchange file) and postprocessors (import exchange file). Syntax and structure analysis only applies to preprocessor testing. Semantic analysis applies to both preprocessor and postprocessor testing. Figure D.1 presents an overview of the preprocessor and postprocessor conformance test process using the implementation method defined in ISO 10303-21 or ISO 10303-22.

In the test process, an Implementation Under Test (IUT) is supplied an input instance of the information model in one format and is expected to translate it into another format. The information model is defined by the relevant conformance class of the application protocol.

For a preprocessor, the specific format of the input is not defined by ISO 10303. For conformance testing purposes, it is defined as a form of human-readable text and graphics ("hardcopy") that represents an instance of the information requirements given in clause 4 of an AP. The preprocessor output format is an ISO 10303-21 exchange structure or a series of ISO 10303-22 calls that provide a model description.

For a postprocessor, the situation is reversed. The input format is specified as an ISO 10303-21 exchange structure or a series of ISO 10303-22 calls for model construction, while the specific syntax and structure of the output is undefined by ISO 10303. The semantics of the output, however, must match those encoded in the input as defined by the AP. For conformance testing purposes, postprocessor output is a series of responses to human-interpretable queries about the semantics contained in the input model instance. Though a postprocessor may provide hardcopy output, this is not assumed or required.

Annex E
(informative)

Revision History

This document may be immediately implemented by any project interested in doing so. The use of this edition is mandated only for documents submitted for stage 30 ballot one year after the publication of this standing document by SC4. Projects that have not yet produced a stage 30 document at the time of this documents approval by SC4 are strongly encouraged to use this edition.

Projects using this edition of the document must use this edition alone, and in its entirety.

Bibliography

[1] ISO 10303-201:1994 *Industrial automation systems and integration — Product data representation and exchange — Part 202: Application Protocol: Explicit draughting*

[2] ISO 10303-203 1994 *Industrial automation systems and integration — Product data representation and exchange — Part 203: Application Protocol: Configuration controlled 3d designs of mechanical parts and assemblies*

[3] BARNARD FEENEY, Allison (ed). *Issues Log - Standing Document Ballot of the Guidelines for the development of abstract test suites, edition 2*, ISO TC 184/SC4/QC N133, 2002-02-20. Available from the Internet: <http://www.nist.gov/sc4/wg_qc/qc/n133/qcn133.htm>.

[4] *Cumulative list of sc4 resolutions,* ISO TC184/SC4 N1218, 2001-10-15. Available from the Internet: <http://www.nist.gov/sc4/ndocs/n1218/n1218.pdf>.

[5] *Guidelines for the development of abstract test suites, edition 2*. ISO TC184/SC4 draft standing document, N873, 1999-08-10. Available from the Internet <http://www.nist.gov/sc4/howto/methods/ats_ed2/ballot/sc4n873.pdf>.

[6] *SC4 Organization handbook*. ISO TC 184/SC4 N1199, 2001-08-10. Available from the Internet <http://www.nist.gov/sc4/ndocs/n1199>.

Index